TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Desarrollo de un sistema para la gestión de logs de seguridad

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: David García Ontiveros

DIRECTOR: Josep Mª Comas Serrano

SUPERVISOR: Rubén Quesada López

DATA: 19 de juny de 2009

Títol: Desarrollo de un sistema para la gestión de logs de seguridad

Autor: David García Ontiveros

Director: Josep Mª Comas Serrano

Supervisor: Rubén Quesada López

Data: 19 de juny de 2009

Resum

La gestión de logs de seguridad se basa en el cometido de supervisar los registros de las aplicaciones y "appliances" , que se ejecutan en un sistema y tienen como objetivo salvaguardar su información y acceso, para poder observar su comportamiento, predecir y detectar posibles problemas. Una plataforma de gestión de logs de seguridad otorga al sistema un incremento en el tiempo de reacción ante un gran número de ataques y accesos no deseados.

En el mercado de la seguridad informática existen plataformas auditoras capaces de realizar dicha gestión, sin embargo su coste es muy elevado. El objetivo de este trabajo es desarrollar una plataforma capaz de recaptar y auditar de forma centralizada los logs de los sistemas operativos, que es el primer elemento de seguridad dentro de una máquina y generar un reporte clarificador para el administrador, realizando una implementación de bajo coste económico mediante librerías y plataformas de código abierto y/o gratuito. Otros objetivos de desarrollo supeditados a los anteriores son: crear una base de datos con la información de los registros de los sistemas operativos de forma que se pueda acceder a ellos rápidamente, implementar un sistema de búsqueda que actúe como interfaz entre la base de datos y el usuario, independizar la plataforma del sistema operativo para que pueda funcionar en cualquiera de ellos y automatizar el proceso para que éste se realice de forma periódica sin necesidad de la interacción del administrador.

La plataforma ha de ser la base de un sistema de gestión de logs de seguridad para cualquier elemento de la red. Se escoge como lenguaje de desarrollo a Java por su independencia con las funcionalidades del sistema operativo y más concretamente a la tecnología JSP, capaz de aunar código de ejecución (Java) y de visualización HTTP). Para que sea centralizado los procesos serán lanzados mediante formularios que se implementaran bajo un navegador web. Como sistema de almacenamiento se escoge MySQL implementado para varios SO, con gran eficiencia en su uso junto a plataformas Java. La aplicación es capaz de generar reportes gracias a las librerías JasperReports, de uso gratuito, mediante plantillas precompiladas. El motor de búsqueda se

asocia a la base de datos utilizando nuevamente la tecnología JSP y los controladores JDBC que permiten a Java utilizar MySQL, de esta forma la plataforma adquiere uniformidad.

Al ser un trabajo de desarrollo los acontecimientos más destacados sucedieron en el transcurso de éste, como la dificultad para el manejo de grandes cantidades de información que repercutió en la implementación de la plataforma al introducir mecanismos que controlasen el consumo de memoria y adecuaran los procesos para una ejecución progresiva y escalable. También se introdujeron cambios en el procedimiento de ejecución debido al incremento de tiempo obtenido en el desarrollo de tareas, alcanzando un consenso entre eficiencia y no-sobrecarga de recursos.

Title: Developing a system to manage security logs

Author: David García Ontiveros

Director: Josep Mª Comas Serrano

Supervisor: Rubén Quesada López

Date: June, 19th 2009

Overview

The management of security logs is based on the task of monitoring application records and "appliances" that run on a system and aim to safeguard information and access, to observe their behavior, predict and detect possible problems . A log management platform security system provides an increase in reaction time to a large number of attacks and unwanted access.

In the market for computer security auditing existing platforms capable of performing such management, but its cost is very high. The aim of this work is to develop a platform capable of collecting and auditing logs of operating systems in a centralized way, which is the first element of security within a machine and generate a clear report for the administrator, thus making possible the implementation of low cost libraries and open access/free platforms. Other development objectives are subject to the above: to create a database with information in the the operating system records so that you can access them quickly, deploy a search system that acts as an interface between the database and the user, separating the platform from the operating system so that it can operate in any of them and to automate the process so that it is carried out on a regular basis without the need for administrator interaction.

The platform should be the basis for a log security management system for any network element. Java is chosen as development language because of its independence vis a vis the functionality of the operating system and more specifically the JSP technology, capable of combining execution code (Java) and visualization (HTTP). In order to be centralized, processes will be launched using forms that were implemented in a web browser. MySQL has been chosen as a storage system an implemented by various operating systems with great efficiency when used with Java platforms. The application is able to generate reports with JasperReport libraries, free of charge, using precompiled templates. The search engine is linked to the database again using JSP and JDBC technology drivers that enable Java to use MySQL, in this way the platform becomes more uniform.

As it is a process still being developed, the most noteworthy difficulties arose

during its development,such as the difficulty in managing large amounts of information that affected the implementation of the platform to introduce mechanisms that control the memory consumption and to fine tune processes for a gradual and scaled execution. Changes were also made in the enforcement process due to increased time in task completion, thus striking a balance between efficiency and not-overloading resources.

ÍNDICE

INTRODUCCIÓN 1

CAPÍTULO 1. SISTEMAS DE GESTION DE LOGS 3

1.1. Gestión de logs 3

1.2.Logs de seguridad 6

1.3.Logs de sistemas operativos 8
- 1.3.1.Logs de Windows 8
- 1.3.2.Logs de Unix 9

CAPÍTULO 2. ARQUITECTURA DEL SISTEMA 11

2.1. Entorno del trabajo 11

2.2. Necesidades del cliente 11

2.3. Arquitectura del sistema de gestión de logs de seguridad de usuarios 13
- 2.3.1. Centralización de logs 13
- 2.3.2. Audición de logs de seguridad 16
- 2.3.3. Interfaz gráfica 17

CAPÍTULO 3. ALMACENAMIENTO DE LOGS 19

CAPÍTULO 4. TRATAMIENTO DE DATOS 25

4.1. Generación de RAW 25
- 4.1.1. Conexión con MySQL 27
- 4.1.2. RAW y CSV 30

4.2. Tablas de producción para Windows 32

4.3. Tablas de producción para Unix 36

4.4. Exportación de resultados: PDF 38

CAPÍTULO 5. ESTRUCTURA DE LA INTERFAZ GRÁFICA 45

CAPÍTULO 6. AUTOMATIZACIÓN Y BÚSQUEDA 51

6.1. Automatización 51

6.2. Sistema de búsqueda 54

CAPÍTULO 7. SISTEMAS DE MEJORA Y RENDIMIENTO 61

7.1 Sistema de mejora 61
- 7.1.1 Timouts del navegador 61
- 7.1.2 Consumo de memoria 63

7.2 Rendimiento de la aplicación 65
- 7.2.1 Tiempos de ejecución 65
- 7.2.2 Indexado de tablas MySQL 65

7.2.3 Ejecución multi-hilo 66

CONCLUSIONES 69

BIBLIOGRAFÍA 73

INTRODUCCIÓN

En las últimas décadas hemos podido observar un brutal incremento en los sistemas de telecomunicaciones, implementados poco a poco en la sociedad han conseguido conectar a prácticamente todas las partes del planeta de forma consistente.
La conectividad que el fenómeno “internet” ha establecido sobre la población ha resultado en una infinidad de negocios sustentados por dicho vínculo.
La capacidad de establecer un comercio abierto las 24 horas y con sedes establecidas en cada punto de conexión de la red de redes otorga al comerciante facilidades de mantenimiento y reducción de recursos necesarios para llevar a cabo su negocio. Estos negocios, para poder funcionar de forma segura necesitan de plataformas que permitan transacciones de datos o económicas con una estrategia de seguridad TIC (tecnologías de la información y la comunicación) no solo como aspecto indispensable para la fiabilidad del negocio, sino para ejercer un cumplimiento de la normativa legal (LOPD, LSSI, normativa de Basilea…) o la obtención de certificados de calidad como el SGSI.
Los sistemas de seguridad TIC pueden ser implementados por un considerable número de elementos hardware y software, como pueden ser firewall’s, vpn’s, encapsuladotes SSL, sistemas operativos, detectores de intrusión, antivirus, etc y cada uno de dichos elementos genera un reporte de actividades, llamado log, en el cual se traza los movimientos de los sistemas, las entradas y las salidas de la red, los fallos acaecidos, etc. El volumen de logs que puede generar una empresa al cabo del día depende de la magnitud de su infraestructura TIC, por ejemplo una empresa de banca electrónica puede llegar a generar unos 2-3 gigabytes de logs, teniendo en cuenta que normalmente el formato de un log es texto plano (requiere menos espacio en disco que cualquier otro formato) fenómeno que implica una sobrecarga informativa que provoca una desinformación para el administrador el sistema.
La necesidad de gobernar toda esa información es incuestionable ya que permite prevenir cualquier predisposición de ataque hacia el sistema, y si el ataque ha sido ejecutado podremos generar auditorias forenses, llamadas así por ser ejecutadas después del ataque, para determinar las vulnerabilidades y los autores del ataque.
Los sistemas de gestión de logs se encargan de recoger la información de funcionamiento de los sistemas ejecutados, analizarla y actuar en consecuencia, dotando a la empresa de una capacidad de reacción mayor ante cualquier vulnerabilidad o fallo en éstos y favoreciendo el seguimiento y control de cualquier proceso interno.
Por lo tanto estamos hablando de un sistema de prevención, de mejora del tiempo de reacción ante problemas y de mejora estructural continua para el propio negocio.
Un sistema de seguridad TIC lo componen una gran cantidad de elementos, ya sean activos o pasivos, hardware o software, siempre con un registro de actividad diferente, es decir, diferentes logs para cada elemento. Dentro de éste amplio abanico, nos hemos centrado en los sistemas operativos, que son la primera barrera de seguridad para cualquier ordenador. A modo corporativo

los 2 sistemas operativos más utilizados son Microsoft Windows y sistemas basados en la arquitectura Unix, es por esta razón por la que nos centraremos en el estudio y gestión de los logs generados por estos sistemas.
La decisión de centrar la gestión de logs de seguridad de los sistemas operativos es debido a la escasez de herramientas auditoras para estos contenidos, es decir, el propio sistema operativo no realiza ningún tipo de supervisión del contenido del log (como hacen algunos tipos de firewall o detectores de intrusión) y las herramientas implementadas en el mercado resultan costosas y no siempre se adecuan a las necesidades de la empresa.

El objetivo de éste trabajo es implementar una plataforma capaz de auditar logs de sistemas operativos Windows y Unix, capaz de recolectar todos sus logs de forma centraliza y automática, de analizar y formatear el contenido de éstos y crear una base de datos para poder almacenar la información. Una vez poseemos una base de datos comprensible y ordenada, crearemos unas herramientas capaces de generar reportes sobre dicha información, realizando agrupaciones y estadísticas y plasmando el contenido en archivos legibles. Sobre la base de datos se creará un motor de búsqueda para otorgar al usuario la capacidad de encontrar cualquier evento de seguridad producido en sus sistemas de manera rápida y sencilla. Es por tanto un trabajo de filtrado de información, realizando tareas para facilitar el análisis de unos datos extremadamente numerosos que pueden llevar a una desinformación total y absoluta al administrador de seguridad.

De esta manera los capítulos se estructuran siguiendo las pautas de un sistema de gestión de logs convencional. Primeramente se describe la composición de un SGLOG convencional para pasar a una serie de capítulos donde vamos desgranando cada parte fundamental implementada en nuestra plataforma. Finalmente terminamos con una recopilación de las implementaciones forzadas por trabajar en un entorno tan sobrecargado de información.

CAPÍTULO 1. SISTEMAS DE GESTION DE LOGS

1.1. Gestión de logs

Un sistema de gestión de logs es una plataforma de recolección, almacenamiento, y audición de eventos informáticos en su formato nativo, comúnmente llamados logs. El objetivo del sistema es poder resumir la inmensa información contenida en los registros, ahorrando de esta manera espacio de almacenamiento en los discos y presentar los resultados de una forma clara y concisa al usuario, dotando al log de algo que por su naturaleza no tiene: brevedad y entendimiento.

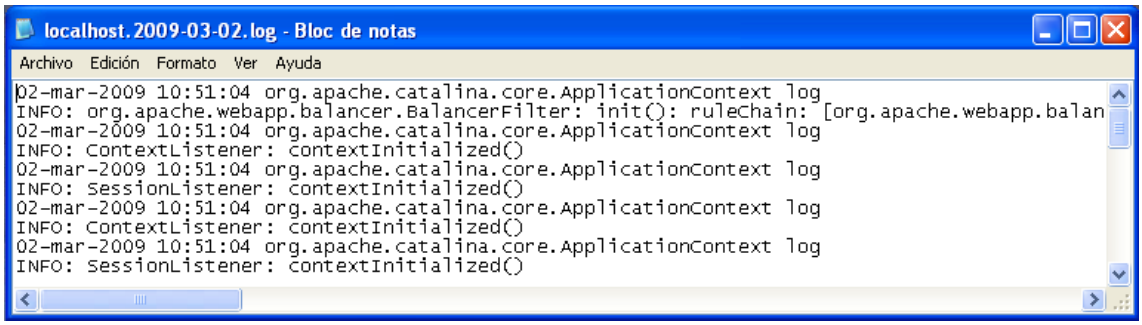

Fig. 1.1 Ejemplo de archivo log

Los logs son archivos de texto llano, como se muestra en la figura 1.1, en los cuales se registran los eventos de cualquier sistema o servicio, entendiendo como evento todo suceso que el creador de la plataforma haya considerado oportuno registrar como entradas y salidas de información, errores de la aplicación, valores de rendimiento, etc. La gran mayoría de herramientas informáticas genera su propio log para el control de su funcionamiento.

Cada sistema de gestión de logs tiene una arquitectura propia y una manera distinta de realizar su tarea, aun así podemos discernir una serie de características comunes:

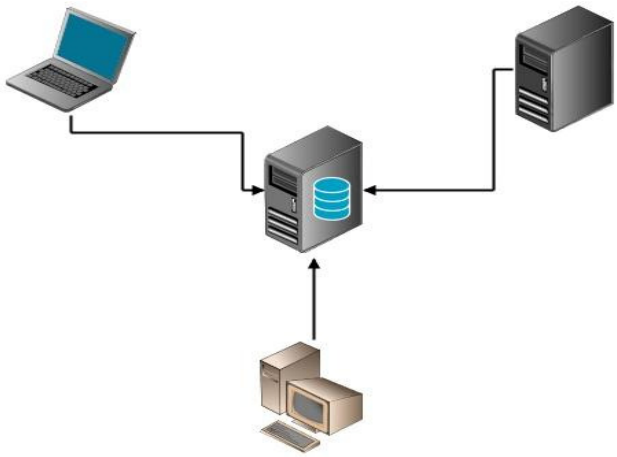

Fig. 1.2 Arquitectura de recopilación de datos

- **Recopilación de datos:** La gestión de los logs se inicia en la recopilación éstos, que puede ser de forma directa (en tiempo real) en la que cada evento generado en un servicio es enviado a la plataforma instantáneamente o indirecta almacenándose los registros en la máquina local y enviándolos al sistema cada cierto periodo de tiempo (diario, semanal, mensual). Los servicios críticos, que han de ser controlados continuamente han de enviar sus eventos de forma directa. Existen plataformas en las cuales no hay la posibilidad de envío directo de los logs, para lo cual se utilizan “agentes”. Estos programas son de reducido tamaño y de carga computacional reducida y que interactúan con el servicio para poder recuperar los eventos y enviarlos a un destino previamente programado. Hemos de tener en cuenta que toda esta información es privada y que toda información privada que viaje por la red debe hacerlo de forma codificada. El incremento de recursos utilizados y el tiempo necesarios para poder hacer esto y teniendo en cuenta que los puntos de envío suelen estar dentro de la propia red local, a la cual se la considera segura, no se realiza encriptación alguna.

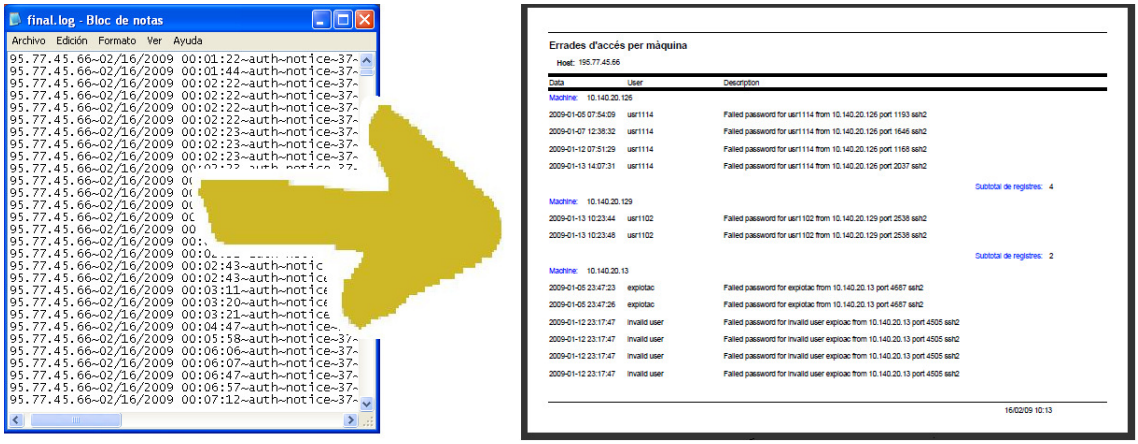

Fig. 1.3 Imagen representativa del objetivo de la aplicación

- Tratamiento de la información: Una vez el sistema obtiene la información nativa, también llamada bruta, se procede a su tratamiento y correlación. Esto consiste en identificar los eventos a auditar, relacionarlos con un tipo de dato o alarma concreta y tomar las medidas establecidas para cada uno. Es evidente, la necesidad de establecer unos valores de medición y unas escalas correctas para poder interpretar de forma precisa toda la información recibida.
Normalmente el efecto obtenido es calificar y clarificar la información, por ejemplo dotándola de un código de colores o parametrizándola para su uso posterior.

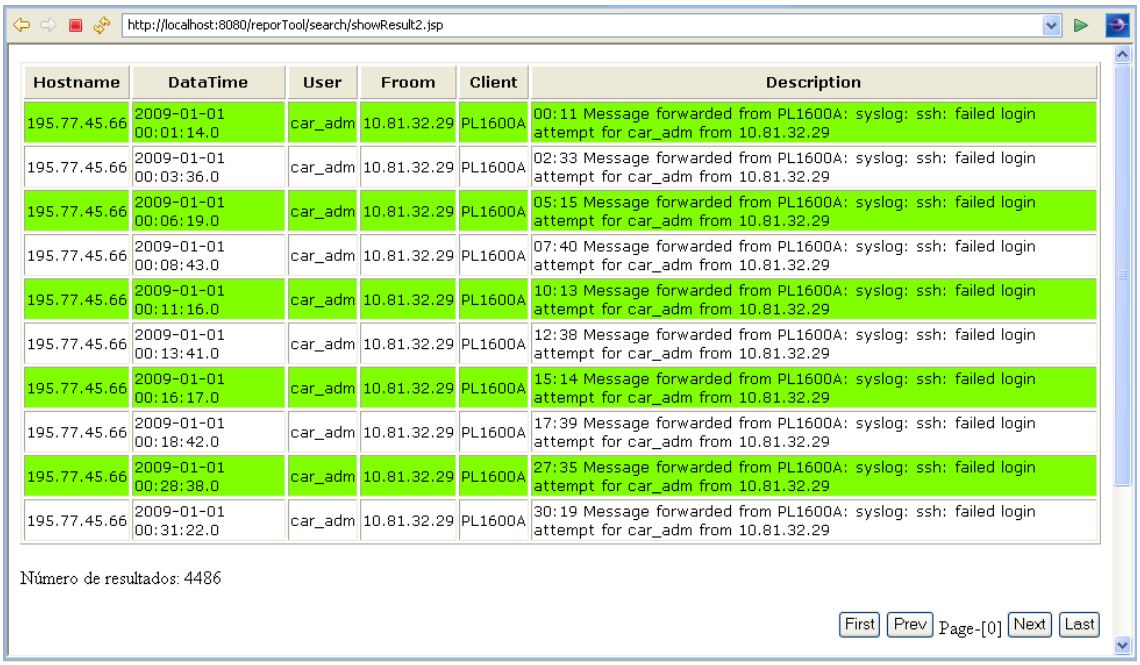

Fig. 1.4 Imagen del sistema de búsqueda.

- Cuadro de mando: La gestión de logs ha de mostrar un resultado sin exceso de información al usuario y de forma centralizada, con lo que un aspecto común es la implementación de un cuadro o consola de mando mediante la cual se pueda configurar y controlar todo el sistema. Mediante dicho cuadro el usuario ha de ser capaz de generar los reportes, de visualizar los eventos auditados y de controlar los agentes o la configuración de audición de los sistemas que envían los eventos.

Actualmente los sistemas de gestión de logs forman parte de plataformas de seguridad “totales” usadas para el control y perimetraje de la red en medianas y grandes empresas. Dichas plataformas soportan aplicaciones de control de puertos, antivirus, antispam así como utilidades que van más allá de la simple gestión como plataformas capaces de realizar audiciones forenses en función del ataque reportado por el administrador.

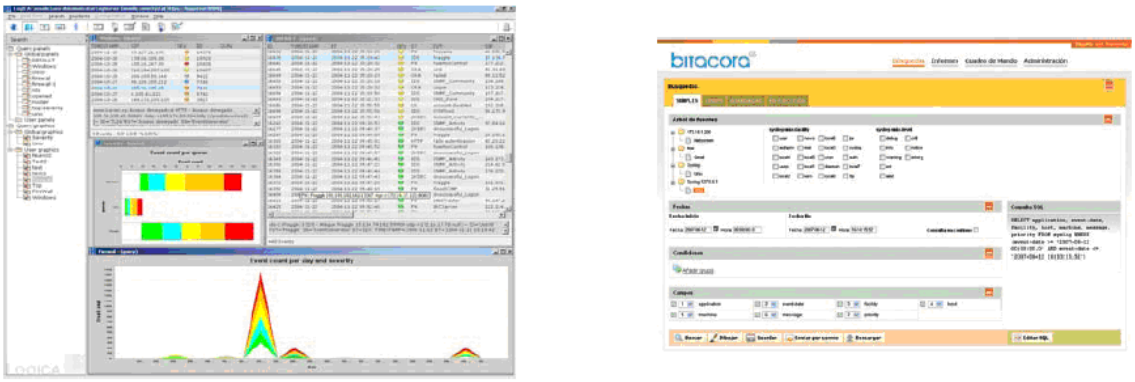

Fig. 1.5 Aplicaciones de seguridad totales

Ejemplos de plataformas tan completas las encontramos en productos como Bitácora de S21sec, figura 1.5-B, que incorpora un amplio abanico de de reconocimiento de diferentes logs, excluyendo el uso de agentes, un sistema de almacenado nativo, es decir, sin base de datos se accede directamente al log que ha sido indexado previamente y un motor de correlación de alarmas y eventos controlable desde un cuadro de mando http.

LogICA del grupo ICA también es un ejemplo de dichas plataformas, figura 1.5-A, basada en agentes y un motor gráfico que resume en tiempo real las estadísticas de eventos del sistema, además contiene un módulo de gestión de evidencias creado especialmente para los análisis forenses.

1.2.Logs de seguridad

Para poder entender la funcionalidad y las características de los logs de seguridad, procederemos a definir diferentes parámetros implicados:

Plataforma de gestión de logs: El objetivo de la plataforma es recoger, almacenar, analizar y reportar una serie de eventos producidos por la ejecución de los diferentes servicios dentro de una estructura empresarial. Estos eventos son producidos por los sistemas que velan por la seguridad de la empresa.

Seguridad informática o seguridad IT: consiste en asegurar que los recursos de los sistemas sean utilizados exclusivamente por las personas acreditadas para ello y dentro de un contexto de privilegios específico para cada uno.

Seguridad: es un estado que indica que el sistema esta absento de peligro, riesgo o daño, entendiendo como tal toda aquella acción que pueda afectar a su funcionamiento o a los datos procesados por este.

Log de seguridad: es un archivo en el cual se almacenan los eventos de seguridad del sistema que lo genera.

Evento de seguridad: es aquel que reporta un comportamiento de acceso a cualquier tipo de información confidencial, ya sea de forma directa, como las entradas y salidas de usuarios en una red local, o indirecta como el funcionamiento de servicios que dispongan de privilegios para manipular datos privados. El reporte de eventos de seguridad concierne a todo aquel sistema que contenga información confidencial o vulnerable.

Todos los logs de sistemas encargados de proteger el acceso a red o a máquinas locales son logs de seguridad, como los cortafuegos, proxys, servidores de autenticación, de dominio, sistemas operativos, etc.

Existen dos tipos de seguridad dentro de una red local: la perimetral y la seguridad interna. La primera, la seguridad perimetral concierne a todos los dispositivos y sistemas de control de acceso desde el exterior. La segunda,la seguridad interna, implica a todos aquellos procesos que verifican la autenticidad de los usuarios y sus posibilidades de operatividad dentro de la red, comúnmente llamados privilegios administrativos.

Los sistemas operativos son responsables de la seguridad interna, ya que contienen el sistema de privilegios y usuarios de la propia máquina, es más, hay servidores que asumen el control de usuarios de todo un dominio o red mediante su sistema operativo.

Para poder entender la importancia de la labor que desarrollan los sistemas operativos mostraremos una analogía:
“Imaginemos un torno de acceso en la entrada de un edificio en el cual introducimos nuestra tarjeta de identificación y un PIN secreto.”
- El torno registra en un documento todas las entradas y salidas de los inquilinos, pudiendo discernir a que plantas tienen acceso, los intentos de entrada erróneos, los cambios de PIN, los cambios de niveles de acceso, etc.

- Si ocurriese un robo en ese edificio, ¿no sería un documento esencial los registros del torno?
- ¿analizar el documento de forma periódica no ayudaría a identificar posibles comportamientos sospechosos?

Los logs de seguridad generados por los SO, concretamente los eventos de gestión de usuarios, tienen la misma importancia, que el sistema de seguridad análogo, en la prevención de ataques informáticos o en el análisis forense del sistema, si el ataque ya ha sido realizado.

Al controlar un punto interno de acceso detectamos intrusiones que hayan podido pasar desapercibidas para los sistemas de seguridad perimetral o incluso ataques de los mismos miembros de la red. Pero estos registros no dejan de ser logs donde existen innumerables acciones positivas, acordes con las políticas del sistema que lo único que muestran es un comportamiento correcto, de ahí la necesidad de analizar en profundidad esos logs y realizar un seguimiento estadístico de los comportamientos para extraer tendencias anómalas.

1.3.Logs de sistemas operativos

Cada sistema operativo reporta de manera propia los eventos producidos durante la ejecución. Existe un gran número de sucesos de seguridad bajo la responsabilidad del SO y hemos de tener la capacidad de decidir cuáles son auditados, mediante la configuración de políticas. En el contexto de un SO tenemos 7 grandes grupos de sucesos de seguridad:

- **Acceso al servicio directorio:** eventos relacionados con servicios de recursos compartidos.
- **Cambios de directivas:** son los producidos cuando se modifican las políticas de auditoria o los derechos de los usuarios.
- **Seguimiento de procesos:** todo lo que concierne a la ejecución de los procesos que corren por el SO.
- **Uso de privilegios:** audita el uso que los usuarios hacen de sus privilegios para modificar el sistema.
- **Administración de cuentas:** controla el cambio, eliminación o desactivación de cualquier cuenta del SO.
- **Inicio de sesión:** registra un evento cada vez que un usuario inicia, cierra sesión o realiza una conexión de red.
- **Inicio de sesión de cuenta:** registra el suceso de un usuario al iniciar o cerrar sesión desde otro equipo.
- **Sucesos del sistema:** controla el cierre o reinicio del sistema cuando éste afecte a alguna política de seguridad.

El paralelismo comentado anteriormente hacía referencia a los eventos de control de entrada y salida del sistema, es decir, administración de cuentas, inicios de sesión y uso de privilegios.

En el mercado empresarial de los SO para equipos no-cliente, los que son usados para el soporte de aplicaciones y servicios "*standalone*", existe un duopolio formado por Microsoft Windows y los sistemas Unix, siendo muchos de éstos últimos distribuidos bajo licencia GPL. Por lo tanto con el fin de concretar un marco de trabajo, se han elegido estos 2 grupos de sistemas operativos como objetos de análisis de la plataforma de reporte de eventos de seguridad:

1.3.1.Logs de Windows

Los sistemas operativos Windows reportan sus eventos mediante funciones alojadas en su núcleo, con lo que hay pequeñas variaciones de procedimiento si consideramos distintos núcleos.

Para tener una idea concreta explicaremos el funcionamiento de reporte de eventos del kernel 5.* (Windows Server 2000, 2003 y Windows XP) que es el más usado por las empresas.

El sistema de "*logging*" de Windows se sustenta del sistema de registro, llamado "*Registry Key*" que no es más que una base de datos donde se

almacena configuraciones y opciones del SO. El sistema operativo almacena los eventos dividiéndolos en 6 categorías distintas:

- **Aplicattion:** son todos los eventos reportados por las aplicaciones que corren sobre el sistema.
- **System:** eventos producidos por la ejecución del kernel.
- **Security:** eventos de seguridad y privilegios.
- **Directory Service:** eventos producidos en el directorio activo de Windows si el host trabaja como controlador de domino.
- **File Replication:** eventos de control de archivos si Windows trabaja como controlador de dominio.
- **DNS Server:** eventos del servicio DNS, si el host trabaja como tal

Los eventos tienen una estructura de campos, en los cuales se recoge el contexto del suceso y un último campo en el cual se describe el propio evento. El contexto viene definido por campos como la fecha, el equipo donde se produjo, usuario que tenía la sesión iniciada o el proceso generador. Cada evento esta clasificado por su importancia, para poder identificar rápidamente su gravedad, siendo la escala la siguiente: Info-Warning-Error | Auditorias de aciertos-errores, siendo las dos últimas exclusivas de los registros de seguridad.

Todos los eventos tienen un identificador numérico que se registra en uno de los campos del registro y que sirve para identificarlo de manera única. Microsoft se encarga de actualizar este identificador para nuevos eventos y proporciona a sus usuarios una base de datos con todos los eventos “oficiales” y una extensa explicación de su contenido.

Cuando un servicio/aplicación quiere registrar un evento utiliza funciones proporcionadas por el kernel para implementar y registrar el evento. Mediante el visor “*EventVWR*” podemos observar los eventos producidos por el SO sin posibilidad de enviar éstos registros a cualquier otra máquina.

1.3.2.Logs de Unix

La familia de sistemas operativos UNIX se caracterizan por ser la evolución del sistema operativo creado por los laboratorios Bell de AT&T en 1969, con lo que comparten muchas características de funcionamiento, como el sistema de registro de eventos.

La responsabilidad del registro de eventos pertenece al demonio “*syslog*” que es un proceso ejecutado en la carga del sistema operativo, llamado “demonio” y que trabaja en segundo plano, sin entorpecer al resto de procesos.
Syslog se encarga de recoger los datos de los eventos, estructurarlos, empaquetarlos de forma definida y enviarlos al destino configurado. El demonio realiza sus funciones mediante un socket: /dev/log y un archivo de configuración: syslog.conf.

Cuando un proceso desea registrar un evento, envía los datos del suceso acaecido al socket de syslog. Una vez recibida la información, *syslogd* consulta el fichero syslog.conf para decidir como tratará la información obtenida.

El fichero de configuración, normalmente, está alojado en la ruta absoluta /etc/ y su estructura se fundamenta en 2 sectores: tipo de registro – ubicación donde se ha enviar. La identificación del evento consta de:
- Nombre de la categoría del recurso del sistema que lo ha generado, como por ejemplo, demonios, mensajes de kernel, sistema de impresión, siendo los de seguridad llamados “auth” y “auth-priv”.
- La severidad del evento, siendo la escala: debug-info-notice-warn-err-crit-alert-emerg.

El identificador es codificado numéricamente para formar lo que se conoce como prioridad. Los posibles destinos de los eventos pueden ser dispositivos de visualización (impresoras, pantalla, etc), otras máquinas, ficheros o pipas (sockets).
La estructura de los paquetes está claramente definida, como hemos mencionado antes y consta de las siguientes partes:

- **Prioridad:** número de 8 bits de identificación de evento.
- **Cabecera:** donde se indica la fecha del evento y el nombre de la máquina que lo ha producido.
- **Texto:** muestra el cuerpo del evento.

La longitud nunca será superior a 1024 bytes, el puerto de envío será el 514 y el protocolo utilizado UDP.

CAPÍTULO 2. ARQUITECTURA DEL SISTEMA

2.1. Entorno del trabajo

Con el objetivo de poner en contexto el presente trabajo, hemos de hablar del proceso de gestación del mismo.

El presente trabajo, és el resultado obtenido del desarrollo de una plataforma demandada por la empresa El cliente a la consultoría LineCom Networks S.L.
La arquitectura, implementaciones y restricciones del sistema nacen, por lo tanto, de las necesidades de la compañía demandante y de la filosofía de los sistemas de gestión de logs de seguridad, presentados en el capítulo 1.
El trabajo pretende plasmar las expectativas demandadas por la empresa cliente, aunque hemos de tener en cuenta que LineCom S.L., entiende la plataforma como un gestor de logs de seguridad para cualquier tipo de hardware o software. Por lo tanto el proyecto general dispone de más entidad de la que se presenta en el trabajo final de carrera..

2.2. Necesidades de El cliente

A continuación presentaremos las necesidades a resolver presentadas por El cliente y la propuesta de objetivos que finalmente se llevará a cabo.

El cliente requiere el desarrollo de una plataforma de trabajo que solvente una serie de necesidades generadas por la acumulación de logs de sus sistemas operativos. Más concretamente, el departamento de informática requiere de un sistema capaz de auditar los eventos de seguridad para la gestión de usuarios mediante los logs generados por todos sus sistemas operativos.

El problema a solucionar es el siguiente: los logs de los sistemas operativos contienen información de todos los eventos sucedidos durante su ejecución y estos se generan de forma local en todas sus máquinas, por lo tanto:

- Extracción de forma automática de los eventos de seguridad de todas las máquinas auditadas, concretamente los de gestión de usuarios.

- Agrupar y formatear la información útil y presentar los datos de una forma clara y comprensible.

- Generación de reportes mensuales de seguridad en formato imprimible.
- Almacenar, a través de una herramienta, los eventos de seguridad de sus sistemas operativos.

- Un motor de búsqueda que permita a los usuarios navegar por la información y encontrar de forma precisa cualquier evento mediante su tipología.

Una vez analizadas las necesidades del cliente, se resuelve la necesidad de implementar un sistema de gestión de logs de seguridad.

Como requerimiento añadido el cliente demanda un sistema capaz de indexar la información extraída para después poder hacer búsquedas desde un entorno gráfico que no supongan un consumo de tiempo elevado. Los resultados obtenidos por la plataforma deberán presentarse en formato imprimible, es decir, mediante un documento de formato conocido y utilizado por la comunidad ofimática.

El cliente también solicita que el entorno bajo el cual pueda correr la plataforma no esté limitado, es decir, que su implementación pueda ser multiplataforma, o como mínimo que pueda ser ejecutado bajo sistemas operativos Windows y Unix.

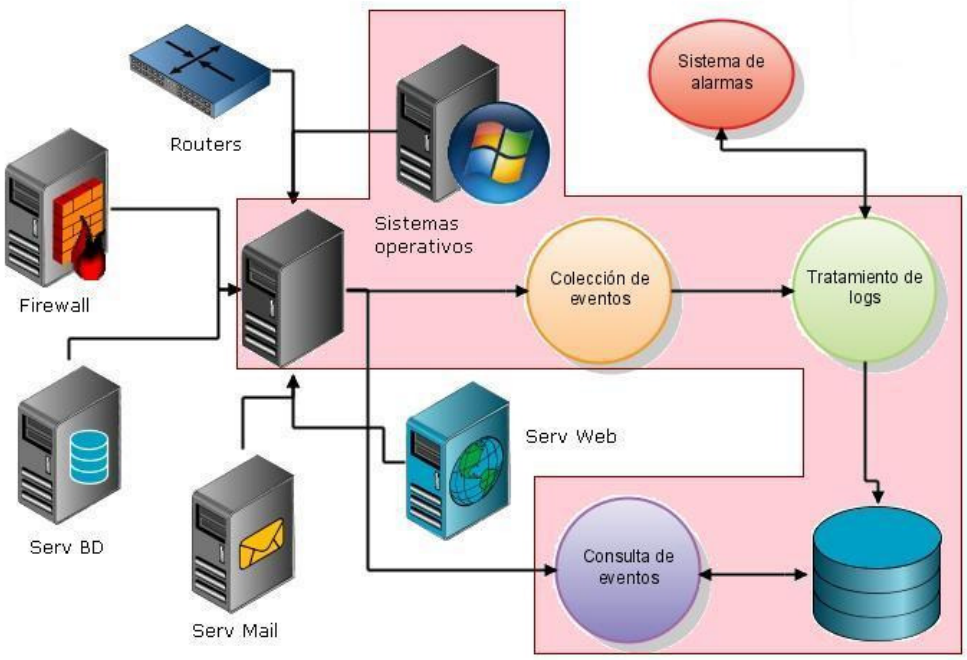

Fig. 2.1 Arquitectura del proyecto Sistema de Gestión de Logs

Como hemos mencionado antes, la plataforma es parte de un proyecto de tamaño superior. Podemos observar en la figura 2.1 el diagrama completo del proyecto, en donde la zona coloreada indica las fronteras de nuestro trabajo.

Existe una recolección de logs de seguridad para una serie de servicios críticos en cualquier negocio y/o empresa, extendiéndose mas allá de los sistemas operativos (y de la gestión de usuarios) y un sistema correlativo de alarmas que trabaja con la información “limpia” de la plataforma. Cabe resaltar que el trabajo

desarrolla la base del proyecto y es totalmente autónomo, no requiere la presencia de los otros bloques de la plataforma para poder realizar sus tareas.

2.3. Arquitectura del sistema de gestión de logs de seguridad de usuarios

La arquitectura está basada en el formato de un sistema de gestión centralizado de logs que ya comentamos en el capítulo 1 Este formato conjuntamente con las necesidades del cliente requiere de una serie de funcionalidades que se traducen en una necesidad de software, deliberadamente libre por razones económicas y arquitectónicas, ya que de esta forma podemos modificarlo para adaptarlo a nuestras necesidades.

A continuación procederemos a desarrollar los requisitos presentados anteriormente.

Requisito multi-plataforma: ya que el servicio ha de poder desarrollar sus funciones en, como mínimo, 2 sistemas operativos diferentes, y teniendo en cuenta que la plataforma de desarrollo ha de ser de distribución gratuita, usaremos el lenguaje de programación Java unido al contenedor de aplicaciones Tomcat.

Java es un lenguaje interpretado, es decir, el código fuente es recogido por una máquina virtual que lo adapta a las funcionalidades del sistema operativo bajo el cual está corriendo.

Por otro lado, necesitamos un sistema de almacenamiento consistente, en el que la información pueda ser organizada, ordenada e indexada para apoyar al servicio de tratamiento de la información. De esta forma hemos escogido la base de datos MySQL de software libre y que ha sido implementada tanto para sistemas Windows como Unix, cumpliendo así con el requisito multi-plataforma.

La base de datos implementada dispone de las siguientes características:

- Es una BD transaccional, asegurando que el fallo de una operación no dañará la información contenida.
- Multihilo, que permite la ejecución de varios procesos en paralelo.
- Multiusuario, que aloja operatividad para varios usuarios de forma concurrente.

Todas las herramientas externas a la plataforma son gratuitas y distribuidas bajo licencia GPL, siendo un requisito, si bien no reclamado por el cliente, si fundamental para la viabilidad y explotación económica del proyecto.

2.3.1. Centralización de logs

El primer objetivo es centralizar los eventos hacia un punto de la red donde se establezca la plataforma.

Cabe resaltar que los sistemas operativos de El cliente son o bien Windows o bien Unix con lo que para el envío centralizamos hemos de tomar distintas estrategias según el SO.

2.3.1.1. Centralización de logs de Windows

Ya hemos comentado como los sistemas Microsoft Windows registran sus eventos y también, de la falta de capacidad para enviarlos a cualquier destino de forma automática. Por dicha razón, utilizaremos un servicio de audición llamado "agente" que interactúa sobre los registros de Windows y nos permite auditar y enviar cualquier evento producido por el sistema en tiempo real.
El software en cuestión es "*Snare Agent*" de la compañía *Intersectalliance* que lo distribuye bajo licencia GPL y se trata de un pequeño servicio que funciona bajo cualquier versión de Windows.

Este programa, nos permite seleccionar los objetivos de audición por categoría, por naturaleza del evento o por su severidad, como muestra la figura 2.2.

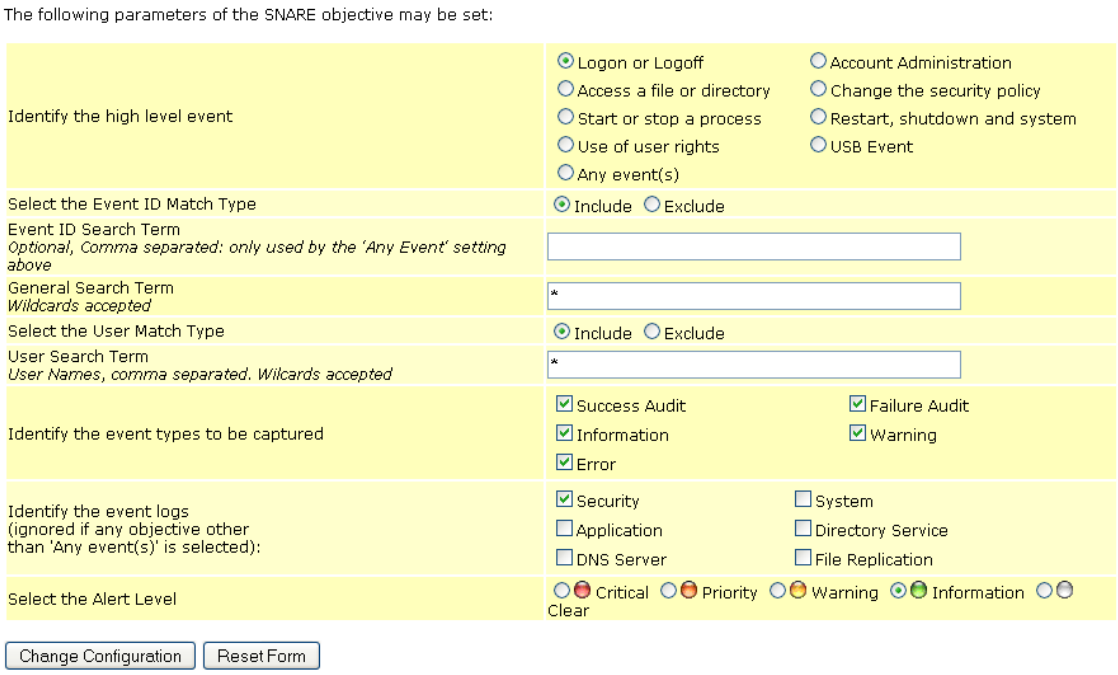

Fig. 2.2 Apartado de selección de eventos del agente SNARE.

Podemos configurar el destino donde se enviarán los eventos auditados, el puerto de envío, el formato de los paquetes y otras opciones de configuración de red, figura 2.3.

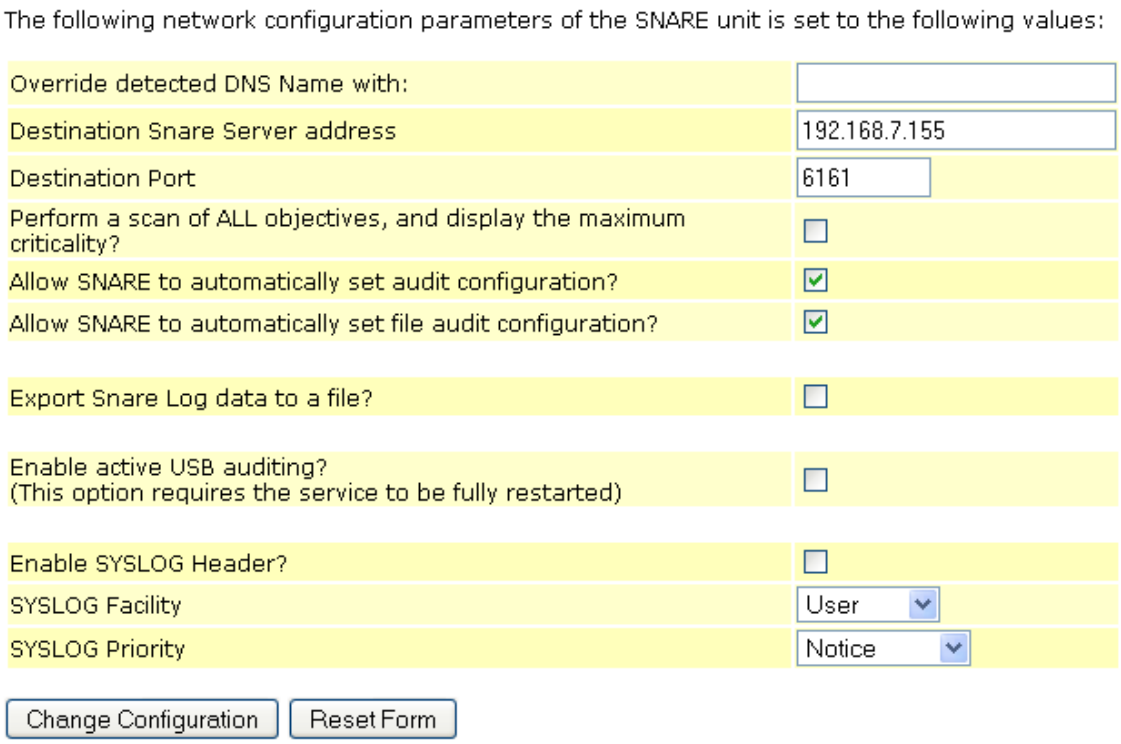

Fig. 2.3 Aparto de configuración de red del agente SNARE

Indicando la dirección y puerto de destino y pudiendo usar el formato syslog para el envío de paquetes.
El agente puede ser controlado remotamente ya que posee un pequeño servidor web que se instala en el puerto 6161.
Mediante el agente podremos enviar cualquier evento de cualquier máquina de Windows hacia nuestro host receptor y de esta manera centralizar los eventos en una sola máquina. Hemos de tener en cuenta que para que el sistema funcione correctamente, el agente se tiene que instalar en todas las máquinas en las cuales corra el sistema Windows y se deseen auditar.
En nuestro caso, como ya hemos comentado, los eventos auditados serán los de seguridad, escogiendo la opción “Security” en el apartado “Indetify the event logs” del agente.

2.3.1.2. Centralización de logs de Unix

Los sistemas Unix poseen la capacidad de auditar y enviar sus eventos a máquinas externas mediante el demonio syslogd.
La única acción que hemos de realizar en las máquinas Unix es identificar el archivo de configuración syslog.conf y editarlo con el fin de indicar al demonio que los eventos de seguridad sean enviados a la máquina recolectora de logs.

Evidentemente, este proceso se deberá de realizar en cada máquina Unix que se desee auditar.

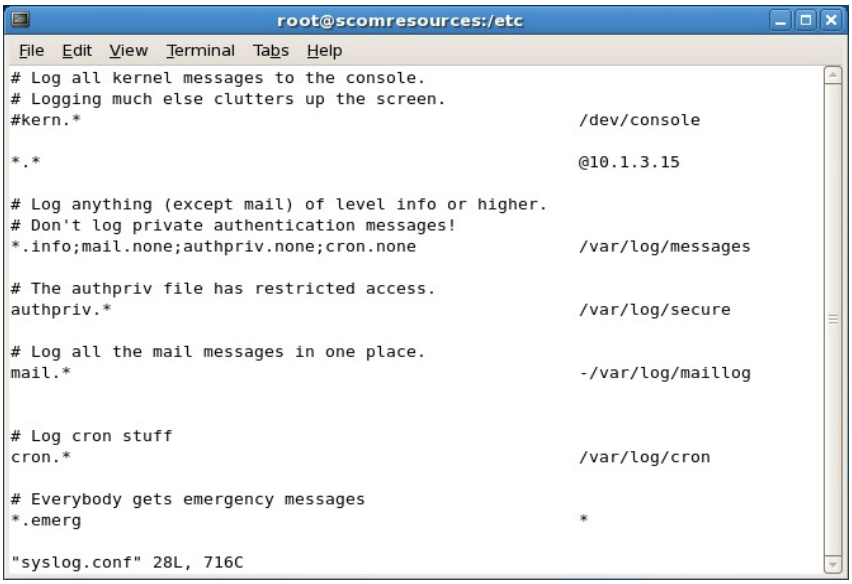

Fig. 2.4 Contenido del archivo syslog.conf

El archivo de configuración tiene un formato sencillo, como podemos observar en la figura 2.4. Al empezar una línea indicamos el grupo de evento a auditar, marcamos con un carácter espacio que ha acabado el objetivo e indicamos la dirección donde se enviar, es decir, escribiremos la siguiente línea:
auth. @ip_host_recolector*
auth-priv. @ip_host_recolector*

De esta forma conseguiremos un envío directo y en tiempo real de los eventos de seguridad del sistema Unix auditado
La explicación del funcionamiento de syslog.conf se puede consultar en el Anexo.

2.3.1.3. Recepción de eventos

Hasta el momento, hemos explicado como los eventos son enviados a una máquina "recolectora", pero en el destino ha de existir un proceso que recoja dichos eventos y los guarde de forma adecuada.
El servicio implementado consistirá en una aplicación "*standalone*" que escuchará por los diferentes puertos de recepción. Cada vez que llegue un mensaje lo guardará en un fichero etiquetado de forma diaria y por máquina de origen.De esta forma conseguiremos crear logs de seguridad y tenerlos clasificados por días y por máquina generadora dentro del host recolector.

2.3.2. Audición de logs de seguridad

Una vez hemos conseguido reunir todos los logs de seguridad de los sistemas operativos necesitamos un servicio capaz de seleccionar los eventos adecuados, estructurarlos e indexarlos para hacer más accesible el contenido y generar reportes mensuales acordes con la información recibida.
Para el almacenamiento estructurado usaremos MySQL como hemos mencionado anteriormente.

A continuación explicaremos alguna de sus características técnicas fundamentales para su inserción en nuestra plataforma:

- Está desarrollado en lenguaje C.
- Utiliza como lenguaje consultor a SQL
- Es posible su uso mediante aplicaciones programadas en casi cualquier lenguaje de programación debido a las API's existentes.
- Proporciona la posibilidad de creación de tablas mediante archivos CSV (comma separated values), que es el tipo de formato usado para el almacenamiento de logs,
- Como plataforma de consulta de datos ofrece la funcionalidad de recuperación de datos de forma ordenada y/o cribada.

El sistema suele trabajar montando un punto de acceso en el puerto 3306/TCP mediante el cual se llevan a cabo las consultas.
Por exigencia del cliente, este proceso será automático y se realizarán actualizaciones de contenido diariamente.

El servicio, usando los datos almacenados y correctamente estructurados, será capaz de generar reportes en formado imprimible usando *JasperReports*, que es una herramienta de código abierto desarrollada íntegramente en Java y que requiere de un editor de plantillas llamado *iReport* que usaremos también en esta plataforma. Seleccionaremos como formato de salida a "PDF" (*Portable Document Format*), ya que es ampliamente utilizado por la comunidad ofimática.

2.3.3. Interfaz gráfica

La aplicación poseerá una interfaz gráfica que permitirá al usuario controlar la información del sistema, generar los reportes consecuentes con la información almacenada y controlar el servicio de recaptación de mensajes.

La plataforma gráfica la ejecutará un servidor Tomcat, que es un contenedor de archivos JSP (Java Server Page).
El servicio gráfico será web para poder ser accesible desde cualquier punto de la corporación sin necesidad de trasladarse al PC o servidor donde está instalada la aplicación. Las páginas JSP son básicamente documentos que contienen código HTML y código Java, debido a éste podemos ejecutar cualquier proceso escrito en este lenguaje desde el servidor y es por eso que necesitamos a Tomcat, para que pueda extraer y ejecutar el código "script" que poseen las páginas. Los "scripts" de las JSP, en nuestra plataforma, no se usarán para generar una interficie gráfica más potente, sino como punto de lanzamiento de los procesos del sistema. Tomcat poseerá todas las funciones productivas del sistema y según elija el usuario las lanzará mediante la máquina virtual, JVM a partir de ahora. La explicación de los procesos de instalación y ejecución de aplicaciones en Tomcat queda relegada al Anexo.
Como requisito del cliente, la interfaz tendrá un apartado para la búsqueda de eventos en su base de almacenamiento, pudiendo exportar los resultados en documentos imprimibles.

CAPíTULO 3. ALMACENAMIENTO DE LOGS

En el capítulo anterior hemos podido ver los recursos necesarios para poder desarrollar una plataforma de gestión de registros, especialmente los que hacen referencia al control de acceso para los sistemas operativos.
A continuación nos centraremos en las herramientas y procedimientos necesarios para la recolección y almacenamiento de los logs.

Una vez los registros llegan a la máquina se han de recoger, identificar y guardar. Para este menester existen varias soluciones de distribución gratuita que se encargan de escuchar por el/los puerto/s y grabar la traza de paquetes en ficheros de texto llano, pero ninguna de ellas cometía con TODAS nuestras necesidades:

- Escuchar por diferentes puertos, no solo por el 514.
- Incluir marcas de tiempo de recepción de eventos cuando éstos sean omitidos en los paquetes originales.
- Clasificar los logs por fecha y origen y guardarlos en ficheros distintos.

Como hemos mencionado en el capítulo anterior, el puerto de envío del agente Snare es configurable, con lo que podríamos enviar los paquetes al 514/UDP, pero se decidió mantener el puerto de facto para poder discernir de manera eficiente los paquetes de sistemas Windows y Unix y tratarlos consecuentemente. De esta manera los logs de Windows y Linux serán enviados a puertos diferentes.

Con tal de poder realizar la identificación de los paquetes que provienen de un sistema o de otro, se decide desarrollar una aplicación con el siguiente modo de funcionamiento:

Syslog envía sus paquetes al puerto 514/UDP y Snare al 6161/UDP, por lo tanto necesitamos un servicio, a partir de ahora llamado eventServer, que entienda el protocolo UDP y que escuche por ambos puertos, siendo éste el que marque si un paquete proviene de un sistema Unix o Windows. Hablamos de servicio porque el proceso ha de estar siempre activo, bajo ninguna circunstancia ha de dejar de funcionar y la mejor forma para que sea automático y transparente al usuario. A continuación especificamos la manera correcta para poder disponer de esta aplicación de manera permanente.

La estructura de eventServer es multi-hilo teniendo como punto de lanzamiento la clase RunServ que únicamente lanza los threads de escucha de ambos puertos. Los threads de escucha se mantendrán en bucle infinito para escuchar constantemente el puerto y delegarán, por cada paquete que reciban, el hilo de ejecución en un nuevo thread que se encargue de procesar el paquete.

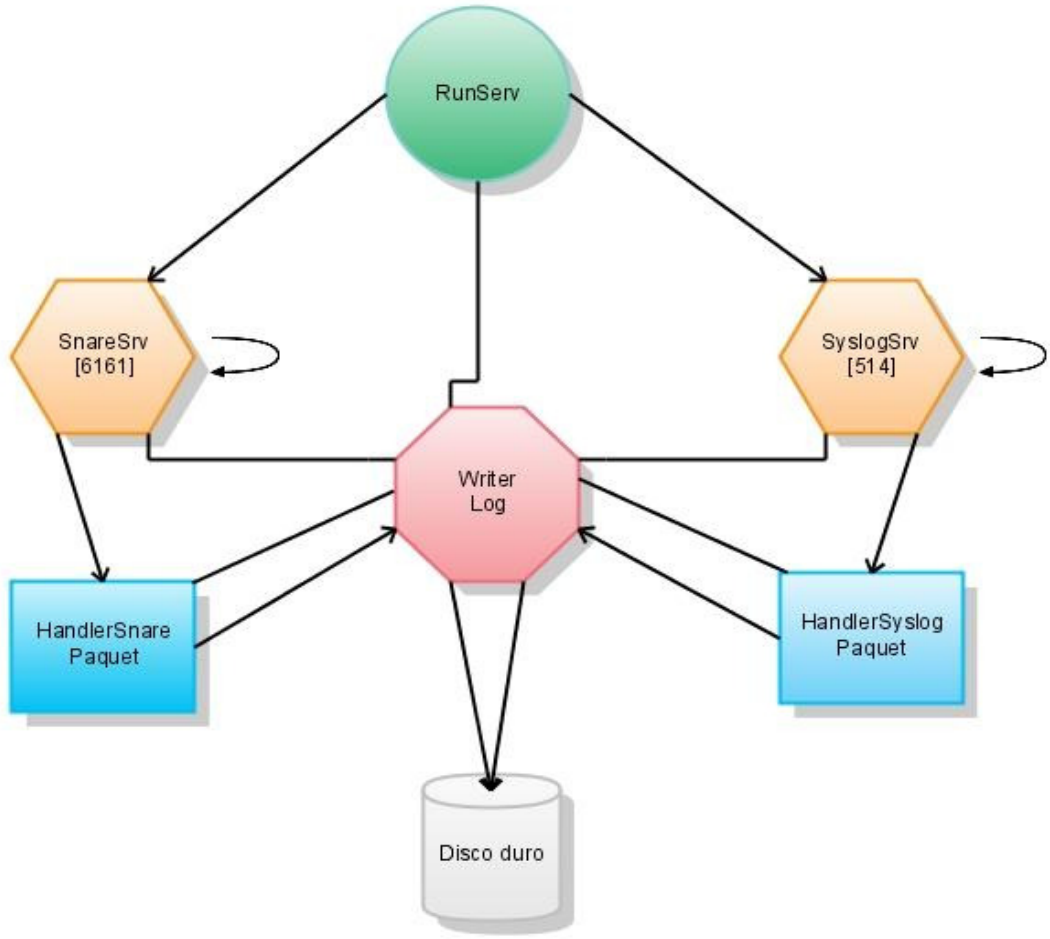

Fig. 3.1 Arquitectura del servidor recaptador de eventos

Los paquetes son almacenados en memoria mediante una instancia de la clase DatagramPacket y es el objeto que se le pasa a los threads “HandlerPacket”. Éste último thread se dedica a procesar los datos, que son convertidos a un formato fácilmente gestionable, como una cadena de caracteres ASCII, para luego analizar su contenido.

El análisis se basa en la estructura de los paquetes Snare y Syslog:

- Snare:

  El formato legible del paquete es el mostrado por la figura 3.2, consta de 16 campos separados por el carácter tabulación. Al final del paquete se incluye un símbolo no legible y a continuación puede contener información de otro paquete, que se usa para comprobaciones del servidor de eventos Snare, que es corporativo y no es incluido en nuestra infraestructura. Así pues el servicio comprueba que el paquete

posea 15 tabulaciones, corta la cadena cuando se han producido y lo guarda en el fichero incluyéndole un salto de línea.

```
ln00005.linecom.local
MSWinEventLog
1
Security
129
Tue Feb 03 12:17:26 2009
861
Security
SYSTEM
User
Failure Audit
LN00005 Seguimiento detallado
El Firewall de Windows detectó una aplicación  al escuchar el tráfico entrante.       Nombre: -    Ruta
```

Fig. 3.2 Contenido de paquete SNARE

- Syslog:

```
<46>syslogd 1.5.0#1ubuntu1: restart.
<86>CRON[6008]: pam_unix(cron:session): session closed for user root/etc/cron.hourly)
<86>CRON[6008]: pam_unix(cron:session): session closed for user root/etc/cron.hourly)
<86>su[5971]: pam_unix(su:session): session closed for user rootroot/etc/cron.hourly)
<86>gdm[5272]: pam_unix(gdm:session): session closed for user linecometc/cron.hourly)
```

Fig. 3.3 Contenido de paquetes SYSLOG

En la figura 3.3 podemos observar 5 paquetes SYSLOG codificados en ASCII. Como podemos ver, la estructura de los mensajes no se ciñe a lo establecido por la norma ya que no disponemos del campo fecha y los caracteres de separación no corresponden a espacios, esto es debido a que el norma describe el formato nativo de SYSLOG que fue implementado para el sistema BSD (antiguo sistema Linux). Por lo tanto, eventServer imprime un timestamp y la IP de origen, extrae la prioridad y la cadena de caracteres informativa, siendo el resultado mostrado por la figura 3.4.

```
2009.02.03 05:16:55     192.168.7.101   46      syslogd 1.5.0#1ubuntu1: restart.
2009.02.03 05:17:00     192.168.7.101   86      CRON[6008]: pam_unix(cron:session): session closed for user root/etc/cron.hourly)
2009.02.03 05:17:00     192.168.7.101   86      CRON[6008]: pam_unix(cron:session): session closed for user root/etc/cron.hourly)
2009.02.03 05:18:20     192.168.7.101   86      su[5971]: pam_unix(su:session): session closed for user rootroot/etc/cron.hourly)
2009.02.03 05:20:15     192.168.7.101   86      gdm[5272]: pam_unix(gdm:session): session closed for user linecometc/cron.hourly)
```

Fig. 3.4 Paquete SYSLOG después de ser tratado por la aplicación recaptadota de mensajes

En el diagrama de la figura 3.1 podemos ver una clase central llamada WriterLog y se encarga de administrar los flujos de datos de la aplicación hacia el almacenamiento en disco cumpliendo 2 funcionalidades:

Se encarga de recoger los mensajes de las excepciones y almacenarlos en un log de funcionamiento de la aplicación y almacenarlos de forma estructurada.

Un aspecto imprescindible de eventServer es almacenar los logs en diferentes archivos clasificándolos por su origen y fecha y esto se

consigue a través la clase WriterLog, que mediante la fecha actual y la IP de origen analiza la ruta de almacenamiento para abrir el fichero adecuado y sino existe lo crea.

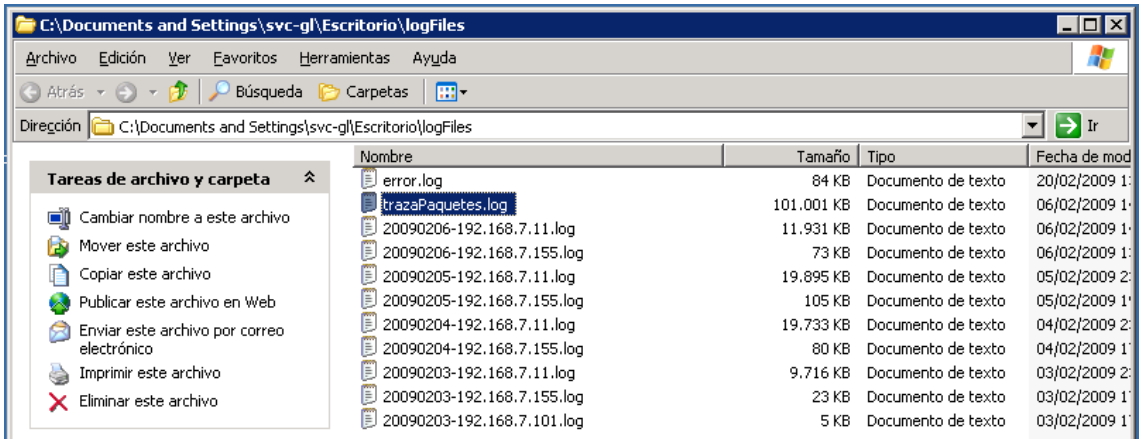

Fig. 3.5 Disposición de logs en la ruta de almacenamiento

Se ha implementado un servicio de conversión de nombres para los logs creados por eventServer. Básicamente, la clase Handler cada vez que le llega un paquete almacena la IP de origen y la compara con una tabla en la base de datos MySQL, si la consulta tiene éxito devuelve el nombre de la máquina, sino será almacenada la dirección IP en el log.

Fig. 3.6 Panel de control de la funcionalidad DNS

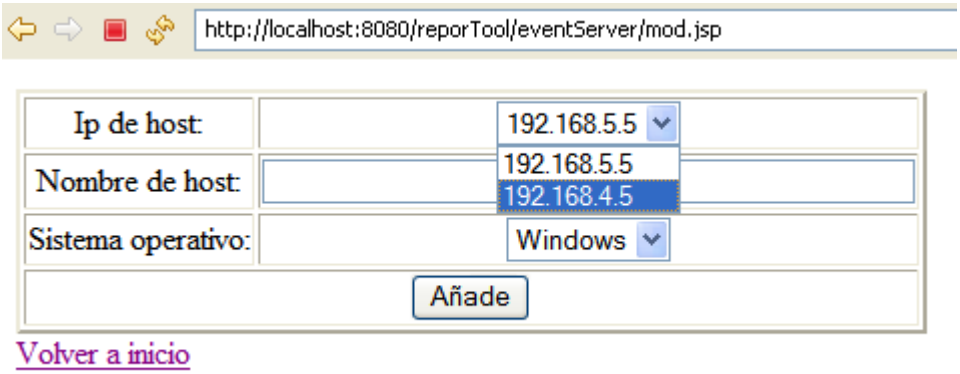

Fig. 3.7 Panel de modificación de la funcionalidad DNS

El entorno gráfico tiene un apartado, como vemos en la figura 3.6, para poder cargar la tabla de convalidaciones: IP = nombre, desde un fichero CVS (con formato específico de separación de columnas mediante un carácter # y separación de filas por salto de línea) y ofrece la posibilidad de añadir, modificar (figura 3.7), borrar o visualizar la tabla de conversiones.
El objetivo de esta funcionalidad es conseguir unos registros con un formato más amigable ya que la traducción de IP's por nombres ayuda al administrador a identificar la máquina del evento con un solo vistazo.

CAPÍTULO 4. TRATAMIENTO DE DATOS

El tratado de datos es el proceso que tiene como objetivo almacenar el contenido de los logs en la base de datos, extraer los eventos auditados, formatear el contenido de las cadenas de datos y guardarlas en una tabla final para el posterior reporte o búsqueda. Por lo tanto estamos en el proceso fundamental de la aplicación, encargado de recoger el “input” y generar el “output” del sistema.

El procedimiento contiene una serie de etapas y conceptos teóricos asociados que explicaremos conjuntamente:

- Generación de RAW’s
- Extracción de eventos
- Almacenamiento de resultados
- Tabla índice

La última etapa no forma parte de proceso en si. Este último apartado corresponde al bloque para el mantenimiento de múltiples tablas en la BD.

4.1. Generación de RAW

El termino RAW corresponde al formato digital de una imagen que contiene la totalidad de de los datos tal y como la lente los ha capturado.
En el mundo informático la palabra RAW se usa para aquel conjunto de información que proviene directamente de la fuente de origen y que necesita ser tratada para su uso de forma cómoda y entendible. Por tanto, un fichero RAW, en el contexto de nuestra aplicación, es una tabla de la base de datos donde se almacena la información contenida en uno o varios logs, esperando para ser procesada y posteriormente borrada.

Para el almacenamiento usaremos tablas de MySQL con formato InnoDB, que es una estructura de almacenamiento que permite transacciones ACID y recuperación por fallos siendo el formato que usa la BD por defecto en detrimento de MyISAM, que si bien es más eficiente en la consulta de datos no ofrece ningún tipo de seguridad en la inserción.
MySQL es incapaz de generar una tabla mediante el formato del log, por lo que primero estudiaremos su formato para crear la estructura de tablas.

NOTA: Recuerda realizar un apartado al principio del documento donde aparezcan todos los acrónimos que has utilizado en el proyecto).

Como hemos mencionado anteriormente existen 2 tipos de logs:

• Windows [Snare]:

Hostname Event Log Type. Criticality. SourceName. Snare Event Counter. DateTime. EventID. SourceName. UserName. SIDType. EventLogType. ComputerName. CategoryString. DataString. ExpandedString.MD5 Checksum

No todos los campos que nos proporciona el log de Snare nos son útiles, pero para automatizar el proceso de almacenamiento en tabla los registraremos todos. A continuación comentamos los más interesantes:

Hostname → Corresponde a la IP de la máquina que genera el evento o el nombre introducido mediante la configuración del agente en la máquina local.
SourceName → Identifica el nombre de la fuente que genera el evento. Este parámetro és extraído del registro de Windows y puede tener los siguientes valores: Applicantion que indica un evento registrado por una aplicación, Security relacionados con la seguridad y System provocados por componentes del SO.

DataTime → Es el timestamp del evento, el instante en el cual se generó.
EventID → Es el ID (identificador) de Windows que posee el evento.
UserName → El nombre del usuario que tiene sesión abierta en la máquina local en el momento de generarse el evento.
EventLogType → Tipo de evento. Natural de los registros de Windows indica la importancia del evento según la escala: información-aviso-error.
ComputerName → Nombre local de la máquina.
CategoryString → Categoría del evento. Natural de los registros de Windows para clasificarlos.
ExpandedString → Donde esta situada la información detallada del evento. En este campo se especifican datos concretos del suceso como pueden ser el nombre del usuario que ha generado el evento y dominio al que pertenece, la dirección IP del origen de la conexión, etc. Es un campo específico de cada tipo de evento y por lo tanto tiene una estructura diferente cada uno.

El parámetro más importante es ExpandedString, ya que es el cuerpo del evento y contiene la información del evento. Mediante el análisis de la cadena lograremos extraer y clasificar la información para poder generar las tablas de resultados.

Una vez analizado el log, concluimos que las tablas necesitaran 15 campos de datos. Todos los campos se definirán como cadena de caracteres, menos los campos EventID, Criticality y Snareeventcount que son siempre número y utilizaremos un integer ya que el identificador de Windows es representado por 4 dígitos decimales.

```
createTableSnareRaw = CREATE TABLE raw_? ( \
  Hostname varchar(512) default NULL, \
  EventLogType varchar(45) default NULL, \
  Criticality int(10) default NULL, \
  SourceName varchar(45) default NULL, \
  SnareEventcount int(10) default NULL, \
  DataTime varchar(45) default NULL, \
  EventID int(10) default NULL, \
  source_Name varchar(45) default NULL, \
  UserName varchar(45) default NULL, \
  SIDType varchar(45) default NULL, \
  EventLog_Type varchar(45) default NULL, \
  ComputerName varchar(45) default NULL, \
  CategoryString varchar(512) default NULL, \
  DataString varchar(20484) default NULL, \
  ExpandedString varchar(2048) default NULL )
```

```
createTableSyslogRaw = CREATE TABLE raw_u? ( \
  DataTime varchar(45) default NULL, \
  Host varchar(45) default NULL, \
  Priority int(10) default NULL, \
  Info varchar(2048) default NULL)
```

Fig. 4.1 Instrucciones para la creación de tablas RAW

• Unix [Syslog]:

Para Unix tenemos 4 campos separados por espacios:
- Data → Es la fecha exacta en la que se produjo el evento.
- Name → IP o nombre de la máquina que ha generado el registro.
- Priority → Número de 2 dígitos decimales que indica la prioridad.
- Info → Cadena de caracteres que explica el suceso.

El campo “Info” es el que contiene los datos para cada evento y por lo tanto deberá ser tratado para la extracción de la información.

4.1.1. Conexión con MySQL

A partir de aquí la aplicación ha de ser capaz de gestionar tablas de la base de datos a través de sentencias MySQL para poder volcar el contenido de los logs en las tablas.

El proceso de vinculación de Java/JSP y MySQL es utilizado asiduamente en las líneas de código de la aplicación, de tal forma se ha estructurado su uso para amenizar la lectura.
La conectividad se realiza a través del conjunto de funciones JDBC (Java DataBase Connection) que permite ligar nuestro código a cualquier base de datos de forma homogénea y transparente.

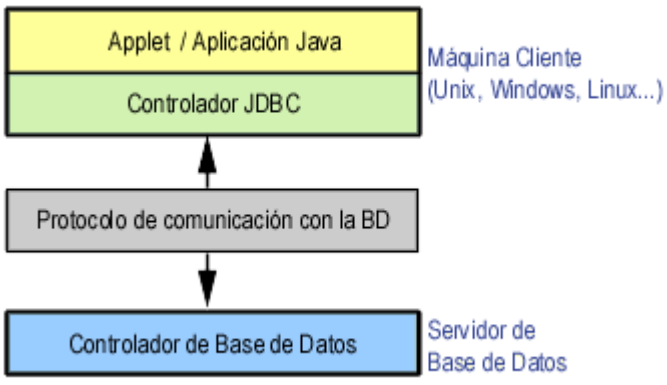

Fig. 4.2 Funcionamiento de JDBC ilustrado en capas de ejecución

Tomcat debe contener los drivers JDBC específicos para poder conversar con MySQL.
Para realizar la conexión sobre la BD se ha creado una clase específica llamada JDBCConnector presentada en la figura 4.3 y que utiliza las librerías JDBC (representadas por el objeto "Connection") y los datos credenciales proporcionados en el contexto Tomcat (variable "envContext") para crear un canal de comunicación con la BD.

```
public class JDBCConnector {

    public Connection createConnection() throws NamingException, SQLException {
        Connection conn;
        Context initContext = new InitialContext();
        Context envContext = (Context) initContext.lookup("java:/comp/env");
        DataSource ds = (DataSource) envContext.lookup("jdbc/testDB");
        conn = ds.getConnection();
        return conn;
    }
```

Fig. 4.3 Clase que hace posible la comunicación con MySQL

La clase busca un archivo de Tomcat, llamado context.xml y que se muestra en la figura 4.4, en el cual hemos especificado los parámetros de conexión, es decir, la URL donde está situada la BD, las credenciales de acceso, tipo de controlador y parámetros de de la comunicación. Se realiza de esta forma la carga de parámetros para que éstos sean accesibles desde cualquier aplicación de Tomcat. En la primera sentencia marcada en el recuadro de la figura 4.3 se especifica donde la aplicación puede encontrar el archivo, en la segunda se realiza la carga de valores de la figura 4.4 mediante el objeto DataSource. Utilizando la función "getConnection()" realizamos la conexión y abrimos el canal de comunicación, pudiendo arrojar excepciones SQL en código si se produce algún error.

```
<Resource name="jdbc/testDB" auth="Container"
            type="javax.sql.DataSource"
            maxActive="100" maxIdle="30" maxWait="10000"
            username="root" password="linecom"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/rtdb?autoReconnect=true"/>
</Context>
```

Fig. 4.4 Contenido de archivo context.xml

La conexión establecida con la base de datos es un parámetro que hemos de tener en cuenta y se tiene que establecer unos protocolos de control. Esto es debido al hecho que la aplicación puede generar gran número de conexiones y si no son bien gestionadas pueden llegar a saturar al servidor de BD por acumulación de reserva de recursos, por lo tanto es importante cerrar aquellas conexiones que no sean necesarias para la ejecución.

Todas las funciones que desempeñan una tarea relacionada con la BD están declaradas en la clase SQLService que comienza declarando 2 variables globales: la ya conocida Connection y un String llamado varSQL donde almacenaremos la sentencia a ejecutar en cada función. Prácticamente todas las sentencias están alojadas en un archivo a parte llamado SQLValues.properties que tiene un formato clave=valor y que mediante la clase ResourceBundle conseguimos extraerlas y guardarlas en la variable varSQL (figura 4.6, bloque try-catch).

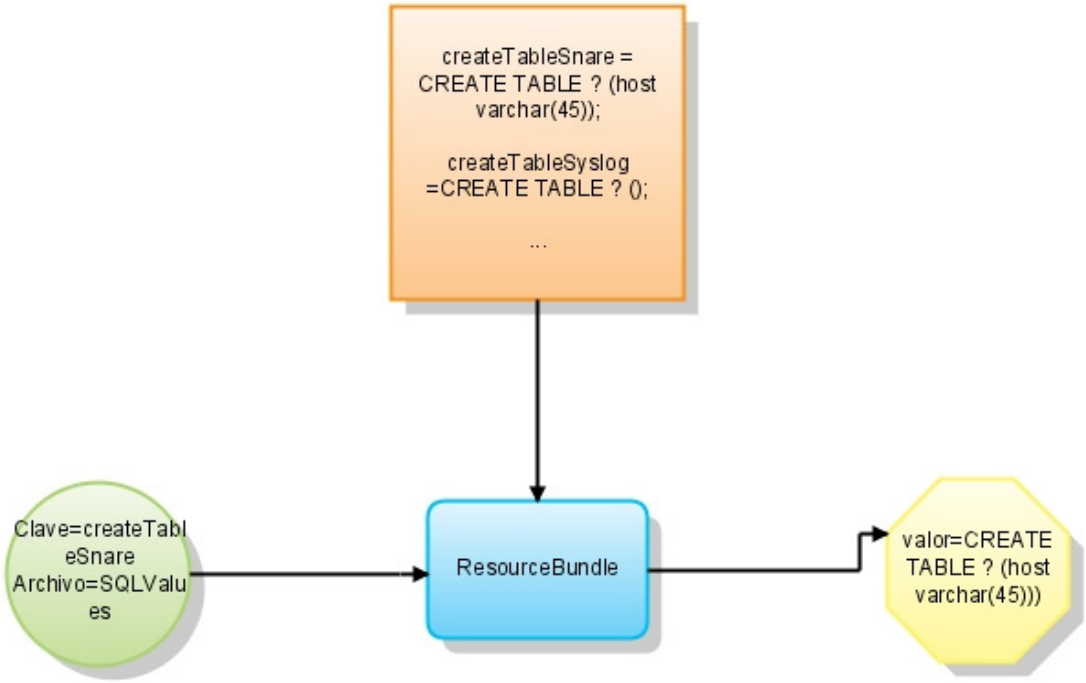

Fig. 4.5 Consulta de sentencias al archivo properties

Una vez tenemos la sentencia, creamos un objeto PreparedStatement que sirve de apoyo para realizar las consultas. Se genera mediante la clase Connection y varSQL como muestra la figura 4.5.

```
// utilizando la clase loaderProps coge valores del archivo especificado
// (sentencia Query)
propertiesFile = new loaderProps();
try {
    varSQL = propertiesFile.loadFieldFileProperties(
            "com.linecom.SQLValues", "loadbyID");
} catch (Exception e) {
}
PreparedStatement pstmt = conn.prepareStatement(varSQL);
pstmt.setInt(1, name);
pstmt.setInt(2, id);

ResultSet dataB = pstmt.executeQuery();
```

Fig. 4.6 Consulta a la BD mediante PreparedStatement

Utilizando PreparedStatement especificamos la sentencia, como podemos ver en la figura 4.6 con la función setInt(), y ejecutamos el contenido. Si la sentencia es una consulta hemos de utilizar la función executeQuery() y hemos de guardar el contenido en una variable ResultSet, si es una modificación hemos de usar executeUpdate() y como resultado obtendremos el número de filas modificadas.
Cuando utilizamos la clase ResultSet hemos de tener en cuenta que no es una variable que guarde sus datos en memoria, sino que es un puntero hacía la BD, con lo que hemos de tratar los resultados y almacenarlos debidamente antes de cerrar la conexión. El enlace será abierto y cerrado en cada función de acceso a la BD al igual que el objeto ResultSet.

4.1.2. RAW y CSV

Para introducir los logs en tablas, lo primero que hemos de tener en cuenta es el formato de los mismos. Podemos observar que cada línea de log presenta los mismos campos, contengan datos o no. Esto es debido a la necesidad de que el log mantenga el formato de tabla con una serie de columnas fijas para facilitar su posible extracción.

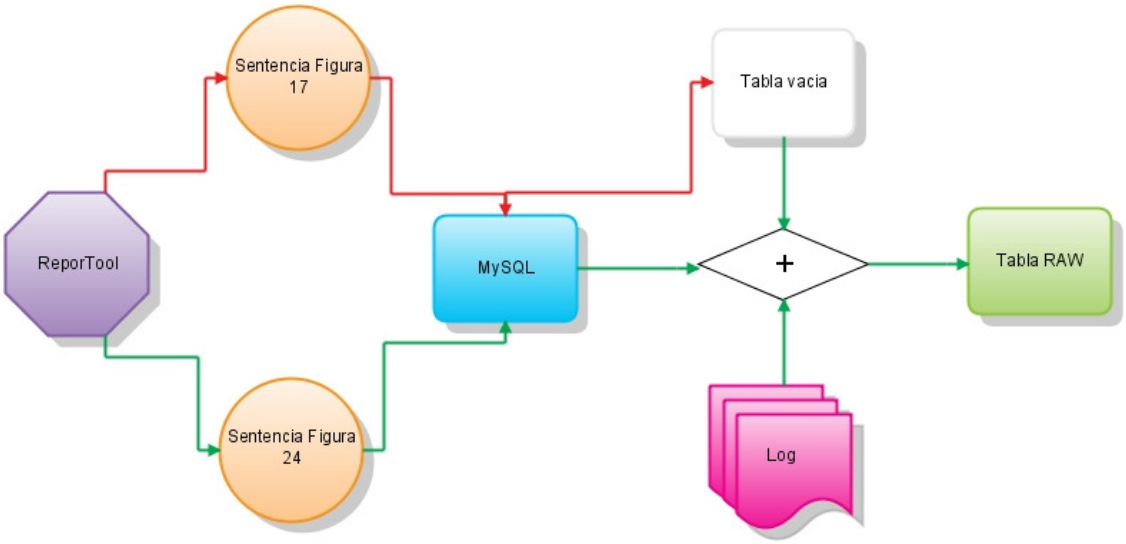

Fig. 4.7 Diagrama proceso de creación de RAW

El proceso comienza con la creación de la tabla en la base de datos, para ello hemos de concretar el formato del log, si es de Unix o de Windows. Una vez creada la tabla mediante las sentencias recogidas en la figura 4.1, proceso indicado en la figura 4.7 por las líneas rojas, se procede a implementarla con los datos del log. Para ello hemos de suministrar la ruta donde se encuentra el archivo y el formato CSV que contiene, indicado en el diagrama de la figura 4.7 por las líneas verdes. Siendo estrictos los logs que recogemos no están dispuestos mediante el formato CVS, que equivale a una estructura de tabla para texto llano en la cual los diferentes valores de las columnas son separados por comas y las filas por saltos de línea, como muestra la figura 4.8.

```
987,juan,87345,10 norte 342
876,pedro,43649,8 oriente 342
123,jorge,03342,av. libertad 23
69,vicente,61560,valencia nº183
18,lorenzo,06490,sol nº 18
19,lucía,06480,luna nº 8
```

Fig. 4.8 Formato original CSV

Es por ello que cuando realizamos la consulta a MySQL le indicamos el fichero, que lo interpretará como CSV, y los caracteres separadores, véase figura 4.9, que en nuestro caso son: tabulación para las columnas. El carácter separador de filas no se lo indicamos puesto que es el mismo que en el formato CSV.

```
fillTableSnareRaw = load data infile ? \
                    into table raw_? fields terminated by '\\t'

fillTableSyslogRaw = load data infile ? \
                    into table raw_u? fields terminated by '\\t'
```

Fig. 4.9 Sentencias para la carga de archivos en tablas MySQL

El carácter “?” reserva el espacio para la introducción de la ruta del fichero y el carácter “\” de final de línea indica se utiliza para poder saltar de línea sin que la sentencia interprete como tal el carácter.
Los símbolos “?” que acompañan al nombre de la RAW reservan el hueco para el identificador, siguiendo la nomenclatura establecida para la plataforma y que analizaremos más adelante.

Este punto de la aplicación es uno de los más críticos del sistema. Esto se debe a errores producidos a la hora de construir las tablas de logs que afectan a la ejecución posterior de las sentencias de MySQL.
Un ejemplo claro que podemos ver es cuando un log no mantiene la estructura de la tabla o contiene mayor información de la que debiera (exceso de campos). En ese caso, cuando MySQL intenta cargar los logs detecta o bien que no hay la información necesaria por línea o que hay excesiva.
MySQL, al tratarse de un sistema transaccional, si detecta un fallo realiza un rollback, es decir, da marcha atrás dejando la tabla tal y como estaba antes de la consulta, vacía. Debido a que trabajamos en un contexto de gran cantidad de datos, donde una carga de un fichero log de 5 GB puede suponer 30 minutos de procesado para un servidor estándar, dichos fallos provocan un gran incremento en el tiempo de ejecución de la plataforma.

Por dicha razón, se implementó una función para testear el formato de los logs antes de enviarlos a la BD. Este sistema de test valida la longitud de las líneas, descartando aquellas que no contengan la información correcta, de esta manera conseguimos que todos los accesos a base de datos a partir de MySQL sean satisfactorios.

4.2. Tablas de producción para Windows

Windows posee una gran cantidad de eventos, todos ellos registrados en una base de datos Microsoft e identificados por un número de forma única. Este atributo es fundamental para poder extraer eventos concretos de nuestras tablas RAW, ya que simplemente especificando el identificador la BD nos devolverá todos los eventos por ser el ID un campo de la tabla. Los eventos escogidos son los demandados por el cliente, todos aquellos que hagan referencia a la entrada y salida de usuarios del sistema, con lo que podemos discernir 3 grupos de eventos:

- Eventos por acceso al sistema: los eventos generados por el SO cuando un usuario intenta acceder o abandonar el terminal. Eventos que van del 528 al 540.

Event ID	Descripció
529	Intent d'accés amb usuari desconegut.
530	Intent d'accés fora del temps permès.
531	Intent d'accés utilitzant un compte deshabilitat.
532	Intent d'accés utilitzant un compte caducat.
533	L'usuari no té permès l'accés.
534	L'usuari ha intentat iniciar un tipus de sessió no permesa, com per exemple una connexió de xarxa, interactiva, massiva, de servei o remota en mode interactiu.
535	La clau d'accés ha caducat
536	El servei d'accés a través de xarxa està inactiu.
537	L'intent d'obrir una sessió ha fallat per altres raons
538	Un usuari ha tancat la sessió.
539	El compte ha estat bloquejat quan s'intentava accedir-hi. Això pot indicar un atac fallit per trencar la clau d'accés la qual cosa ha provocat que el compte fos blocat.
540	Connexió remota iniciada correctament. Això indica que un usuari remot s'ha connectat a un recurs local del servidor, generant un identificador per l'usuari remot.
682	Un usuari s'ha tornat a connectar a una sessió de serveis de terminal. Això indica una connexió prèvia a través d'un servei de terminal.
683	Un usuari s'ha desconnectat d'una sessió a través d'un terminal sense sortir. Això passa quan un usuari està connectat als serveis de terminal remotament. Aquest event es genera en el servidor del terminal.

Fig. 4.10 Eventos de errores de acceso

- Eventos por validación de dominio: si la infraestructura de la empresa dispone de servidores de validación de dominio, estos generan registros por cada usuario logueado en él. Eventos identificados por la serie 672-683.

Event ID	Descripció
672	El servei d'autentificació (SA) ha validat i assignat un tiquet.
673	El servei de concessió de tiquets (SCT) ha assignat un tiquet.
674	Seguretat principal ha renovat un tiquet SA o SCT
675	Preautentificació fallida
676	Petició de tiquet d'autentificació fallida.
677	Un tiquet de SCT ha estat denegat.
678	Un compte ha estat mapejat correctament en un domini.
680	Identificació del compte utilitzada per inici de sessió. Aquest event també indica que el paquet d'autentificació ha estat utilitzat per autentificar el compte.
681	Intent de connexió en un compte de domini
682	Un usuari s'ha reconnectat a una sessió de Servei de Terminal.
683	Un usuari ha desconnectat una sessió de Servei de Terminal sense desconnectar-se.

Fig. 4.11 Errores de validación en dominio

- Eventos por gestión de cuentas: no son rigurosamente eventos de entrada o salida pero si están íntimamente ligados, ya que son eventos generados por el uso de cuentas y éstas son utilizadas para el acceso al sistema. Eventos que forman parte: 624-644.

Event ID	Descripció
624	Compte d'usuari creat
625	Canvi de tipus en un compte d'usuari
626	Compte d'usuari activat
627	Intent de canvi de clau d'accés
628	Clau d'accés creada per un usuari
629	Compte d'usuari deshabilitat
630	Compte d'usuari eliminat
631	Grup Actiu Global de Seguretat creat
632	Usuari afegit al Grup Actiu Global de Seguretat
633	Usuari eliminat del Grup Actiu Global de Seguretat
634	Grup Actiu Global de Seguretat esborrat
635	Grup Actiu Global de Seguretat creat
636	Usuari afegit al Grup Actiu Local de Seguretat
637	Usuari eliminat del Grup Actiu Local de Seguretat
638	Grup Actiu Local de Seguretat esborrat
639	Grup Actiu Local de Seguretat canviat
641	Grup Actiu Global de Seguretat canviat
642	Compte d'usuari canviat
643	Política de domini canviada
644	Compte d'usuari bloquejat

Fig. 4.12 Cuentas de usuario

Como hemos comentado anteriormente, no todos estos eventos nos indican un comportamiento anómalo en la entrada del sistema, si bien nos pueden indicar patrones de conducta aceptados algunos pueden generarse en gran cantidad provocando un sobre exceso informativo. Con lo que hemos realizado una selección de eventos por cada grupo:

- Acceso al sistema → Lo más indicativo para detectar un ataque son los intentos fallidos de conexión local: 529, 530, 531, 532, 533, 534 y 537. Estos intentos pueden deberse a que un usuario ha olvidado su contraseña, con lo que es preciso comparar éstos patrones con otros sucesos inusuales. También puede darse el evento 539 que indica cuenta bloqueada y posible intento fallido contra una clave de acceso si éste va acompañado de eventos 529.
- Validación de dominio → Eventos 675 y 677 que indican intentos de conexión al dominio fallidos.
- Cuentas de usuario → Eventos 644 y 642, el primero muestra una cuenta bloqueada al introducir la contraseña incorrecta varias veces, la segunda es un indicativo del estado: bloqueada, borrada, cambio de políticas, etc.

La información detallada del evento, llamada ExpandedString por el agente Snare, posee una estructura de campos con un formato campo:información de campo y éstos separados entre si por un carácter espacio, como por ejemplo:

- campo1:hola campo2:adios campo3:qtal

El número de campos depende del evento y de la versión del sistema operativo. Para mantener la coherencia de los reportes no es posible separar los eventos de un mismo grupo en tablas diferentes con lo que se ha optado por parsear los campos comunes en los eventos del mismo grupo a auditar.

La estrategia de parseo de los eventos se muestra en la figura 4.13 para eventos de cuentas de usuario. El proceso es el mismo para el resto de grupos, salvo que el campo data se divide en diferentes campos.

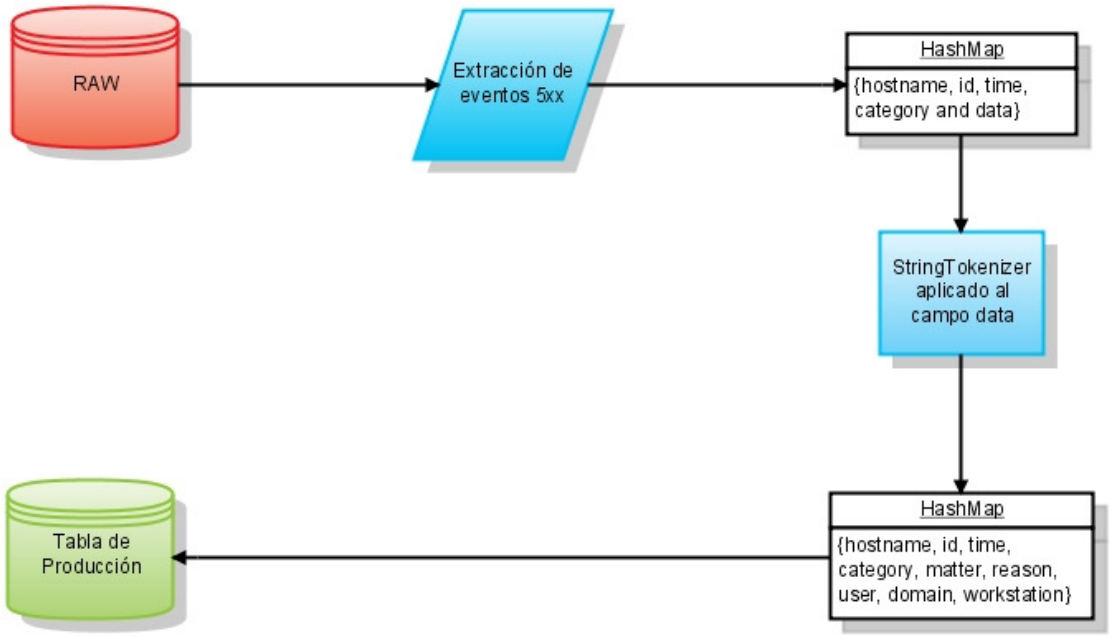

Fig. 4.13 Flujo de parseo evento 529, Windows

La extracción y tratamiento de eventos se realiza por de forma independiente y secuencial, recorriendo cada fila del objeto ResultSet que es el resultado de la consulta a MySQL indicando el/los identificador/es de evento. A continuación se muestra de manera secuencial los pasos a seguir:

- En primer lugar se guardan todos los campos de la fila en un objeto HashMap, que es una lista clave-valor, indicando el nombre del campo y el valor.

- En segundo lugar, del mapa generado se extrae el campo “data” (en la tabla es llamado ExpandedString) y se introduce en una función que mediante el ID parsea el campo data usando la clase StringTokenizer que divide el String según un carácter deseado, mediante 2 bucles uno para separar los diferentes campos y otro para extraer la información de cada campo se consigue generar un nuevo mapa con el campo data dividido en diferentes campos.
- En tercer lugar, el mapa, que siempre contendrá un numero de campos acorde con el identificador de evento al que pertenece, es introducido en una tabla de producción, habiendo sido creada específicamente ésta para dichos eventos. Los campos se van extrayendo del mapa e introduciendo en la sentencia SQL de forma secuencial, para acabar ejecutando la función “executeUpdate()”.

```
ResultSet dataB = pstmt.executeQuery();

while (dataB.next()) {
    HashMap<String, String> mTest = new HashMap<String, String>();
    mTest.put("event.hostname", dataB.getString("Hostname"));
    mTest.put("event.id", dataB.getString("EventID"));
    mTest.put("event.data", dataB.getString("ExpandedString"));
    mTest.put("event.time", dataB.getString("DataTime"));
    mTest.put("event.category", dataB.getString("CategoryString"));

    HashMap<String, String> auxiliarMap = new HashMap<String, String>();

    auxiliarMap = parser.parseRAW(mTest, config);

    if (mMap.get("action").toString().compareTo("0") == 0)
        type = Integer.parseInt(tableTarjet.substring(3, 6));
    System.out.println("El mapa de insertFormatedData: " + auxiliarMap
            + " y el tipo: " + type);
    insertFormatedData(auxiliarMap, type, tableTarjet);
}
```

Paso 1

Paso 2

Paso 3

Fig. 4.14 Código de parseo

4.3. Tablas de producción para Unix

El proceso para generar tablas de producción de Unix es diferente al de Windows por la naturaleza de los logs. En Unix ya hemos comentado que los eventos no son identificados de manera única y es por esto que el proceso se vuelve más complicado.

El primer paso que hemos de realizar es la identificación de los eventos que queremos auditar. Haciendo un símil con los eventos de Windows podemos discernir los 3 mismos grupos:

- Relacionados con el acceso al sistema tenemos 2 eventos: “intento de acceso erróneo” que marca el acceso de un usuario concreto a un sistema y “acceso correcto” que se genera al teclear una contraseña correcta para un usuario concreto, con prioridad 38 = 4x8 [Seguridad o Auth] + 6 [gravedad: información].

- Para el acceso fallido al dominio existe el evento “timeout antes de autorización” con prioridad 34 = 32 +2 [Auth + Crítico] que nos informa que el usuario ha tardado demasiado en validarse en el dominio, es decir, que ha probado varias veces de entrar sin éxito.

- En el grupo de cuentas de usuario podemos distinguir 2 eventos: “cambios erróneos en cuentas” que nos informa de un error al intentar asumir privilegios de root por parte de algún usuario, con prioridad 34, y “password erróneo” que nos informa del intento de acceso a una cuenta con un password incorrecto, con prioridad 38.

Como podemos observar existen varios eventos con la misma prioridad, pero el rango que obtenemos es de dos, 38 y 34. Utilizaremos la prioridad para poder

extraer los eventos, que si bien existen eventos no auditados con la misma prioridad al menos reduciremos el volumen de datos con los que trabajará la aplicación.
Una vez hemos logrado separar eventos por prioridad, el segundo paso es seleccionar los eventos elegidos. Para ello necesitamos acceder a la información del evento y compararla con una máscara para poder identificarlo.

```
if (mode.compareTo("PER") == 0) {
    priority = 38;
    mask = "Failed password";
    position = 0;                          <- Posición en la frase
} else if (mode.compareTo("IAE") == 0) {
    priority = 38;
    mask = "failed login";                 <- Término que ha de coincidir para poder identificar
    position = 7;
} else if (mode.compareTo("TAA") == 0) {
    priority = 34;
    mask = "fatal: Timeout";
    position = 0;
} else if (mode.compareTo("CEU") == 0) {
    priority = 34;                         <- Prioridad genérica del evento
    mask = "BAD SU";
    position = 6;
} else if (mode.compareTo("RAC") == 0) {
    priority = 38;
    mask = "Accepted password";
    position = 0;
}
```

Fig. 4.15 Máscaras Unix para parseo

El proceso a seguir para poder realizar la identificación del evento es dividir el string de información por palabras (con StringTokenizer) y almacenado en orden. De esta forma indicamos la posición de la palabra en la frase y el término que ha de leerse para que el evento sea identificado.

Este es el proceso que se ha de seguir con todos los eventos que contengan prioridades 34 y 38, proceso que supone un aumento considerable de la carga computacional.

Una vez identificados los eventos hemos de almacenarlos en HashMaps de una forma similar al proceso de parseado de Windows.

Para poder realizar el almacenaje, necesitamos generar los mapas a partir de los campos de los diferentes eventos.. Debido a que la información es una frase y no una serie de campos estructurados, hemos de buscar en el léxico los valores que necesitamos extraer. La comparación de eventos iguales acaecidos en tiempo diferente, nos permitirá identificar la estructura que necesitamos. La figura 4.16 muestra dos sentencias del mismo evento, mostrando su estructura similar.

Message forwarded from PL1600A: syslog: pts/41: failed login attempt for UNKNOWN_USER from 141.125.125.170

Message forwarded from PL1600A: syslog: pts/44: failed login attempt for deseprov from pc-des-83

Fig. 4.16 Diferentes sentencias del mismo evento.

De esta forma, habiendo identificado previamente el evento, podemos extraer los campos que nos interesen.

Como paso final, una vez extraídos y guardados en un nuevo HashMap, procedemos a insertar éste objeto como una nueva fila de la tabla de producción, como hemos realizado en Windows, de forma secuencial con el mapa.
La figura 4.17 nos muestra el proceso de parseo para un evento Unix de la clase “password erróneo”.

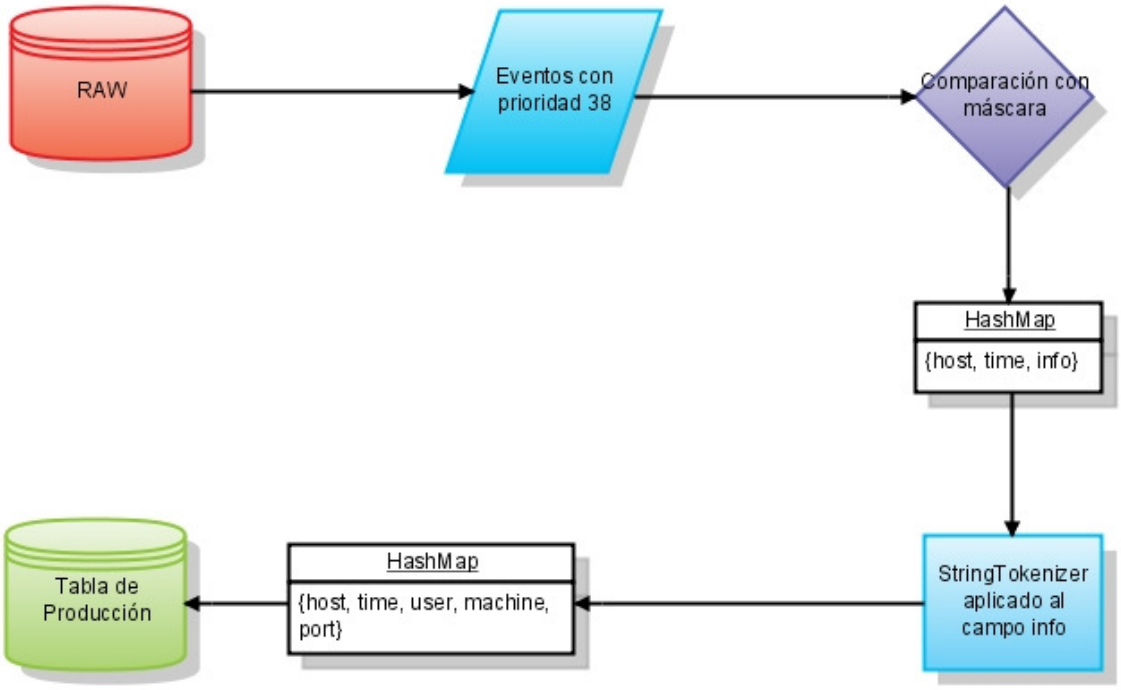

Fig. 4.17 Flujo de parseo PER, Unix

4.4. Exportación de resultados: PDF

Con el mecanismo anteriormente, obtenemos una serie de tablas que contienen eventos del mismo tipo, estructurados en campos de fácil acceso. Para poder identificar los rangos temporales contenidos en las tablas se crea una tabla-índice con información sobre el periodo de tiempo, el sistema operativo y el tipo de evento almacenado (que será especialmente útil para el sistema de búsqueda).

A partir de aquí hemos de transformar dichas tablas en documentos de fácil lectura. Para ello usaremos la plataforma JasperReports, conjunto de librerías para la creación de informes bajo la plataforma Java, con licencia LGPL (código abierto). Hemos de considerar que existe una versión "profesional" con soporte de la empresa responsable de su evolución JasperSoft que a su vez proporciona una interfaz gráfica también GPL, para poder editar plantillas de reportes de forma más intuitiva que utilizando líneas de código, llamada iReport. El proceso para generar un reporte mediante esta plataforma es el mostrado en la figura 4.18.

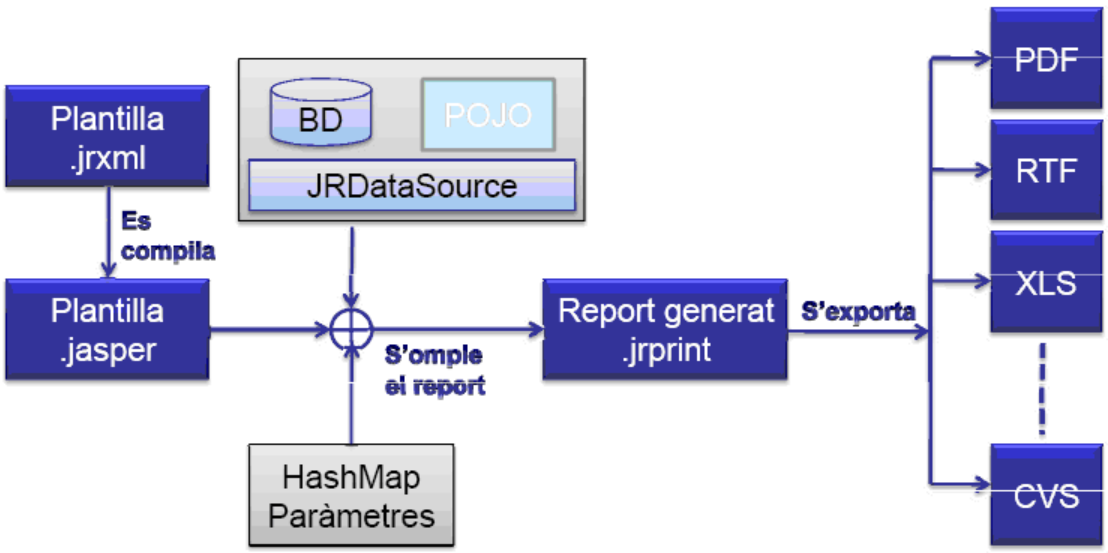

Fig. 4.18 Secuencia para crear un reporte con JR

El proceso parte de una plantilla jrxml, que es un documento con estructura XML que contiene el diseño del reporte y es el resultado del uso de iReport. A grandes rasgos el editor tiene la estructura que muestra la figura 4.19.

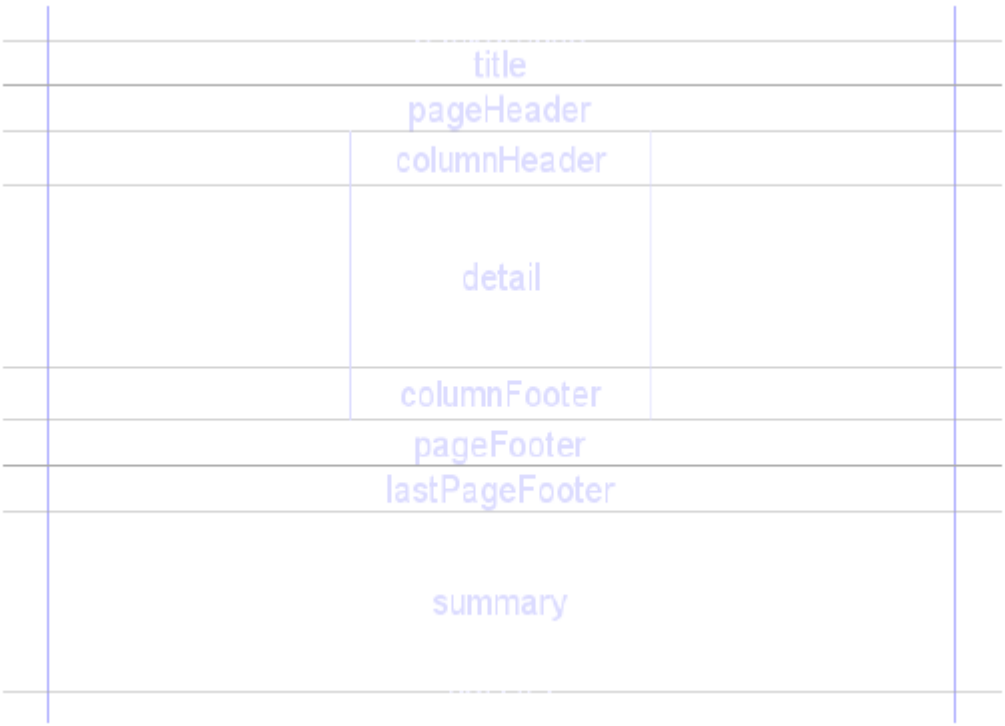

Fig 4.19 Estructura básica de una plantilla jrxml

Cada plantilla de IReport tiene la misma estructura basada en dividir el documento en porciones llamadas bandas. La diferencia entre una banda a parte de su posición espacial es su frecuencia de impresión, por ejemplo, la banda “title” solo se imprime en la primera página del todo el reporte, la banda “pageHeader” en la cabecera de cada página, “detail” es la banda donde se imprime la información siendo el cuerpo del documento, “pageFooter” se imprime al final de cada página del documento, “lastPageFooter” solo en la última página del documento al igual que “summary”.

Existen 3 tipos de datos que almacenan variables:

- “field” recoge valores del DataSet.
- “variable” que son utilizados para almacenar resultados de operaciones dentro del reporte.
- “parameters” que son variables de entrada que no pertenecen al DataSet y son utilizadas normalmente para las consultas a la fuente de datos.

Con DataSet nos referimos a los recursos que contienen la información a reportar, que en nuestra aplicación son las tablas de producción.

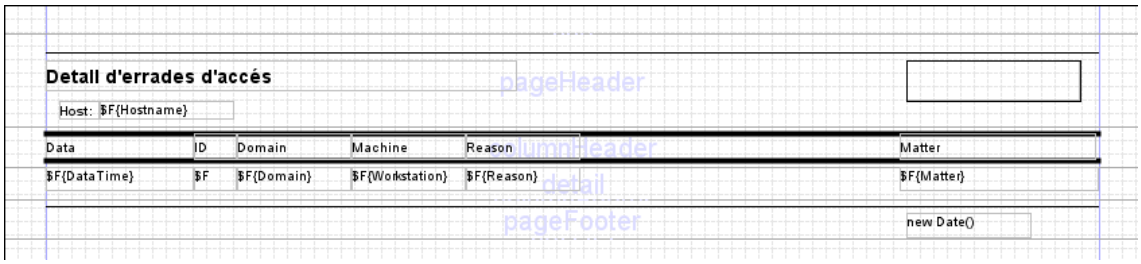

Fig. 4.20 Ejemplo de plantilla jrxml antes de ser procesada.

La generación del reporte es secuencial con lo que hay que ordenar los datos del DataSet para que sean pintados de forma correcta.
IReport también tiene herramientas de dibujo (cuadrados, rectas, etc…), de consultas al DataSet (posibilidad de ordenar los resultados antes de ser utilizados por el reporte), incrustación de imágenes, ayuda para la escritura y otros aspectos comunes de una herramienta de diseño gráfico, viendo un ejemplo en la figura 4.20 de una plantilla para Errores de acceso, es decir eventos 5xx.

Una vez creada de forma manual la plantilla jrxml, que es legible ya que es lenguaje XML, se compila para generar un documento .jasper que contiene los links a las partes específicas de las librerías Java y código no legible que usa el motor de JasperReports. A la plantilla .jasper se le introduce la fuente de datos para llenar el reporte. Los datos externos han de estar configurados previamente en la plantilla para que los acepte, ha de conocer previamente la estructura de la tabla con la que trabajar. El motor de JasperReports crea el documento basándose en la plantilla y la tabla proporcionada y genera un archivo.jprint que es propio de la librería y es un documento acabado pero que solo puede ser leído por el visor del propio motor (JasperViewer). A partir de aquí el jprint puede exportarse a una serie de formatos aceptados por JasperReports como PDF, XLS, RTF, etc. En nuestro caso el formato elegido es PDF, viendo el resultado en la figura 4.21 para eventos 5xx.

Detall d'errades d'accés

Data	ID	Domain	Machine	Reason	Matter
2008-11-16 0:12:58	537		-	An error occurred	Logon Failure
2008-11-16 0:12:59	537		-	An error occurred	Logon Failure
2008-11-16 0:13:0	537		-	An error occurred	Logon Failure
2008-11-16 1:54:2	537		-	An error occurred	Logon Failure
2008-11-16 11:54:27	537		-	An error occurred	Logon Failure
2008-11-16 11:54:28	537		-	An error occurred	Logon Failure
2008-11-16 11:54:29	537		-	An error occurred	Logon Failure
2008-11-16 16:35:14	537		-	An error occurred	Logon Failure
2008-11-16 17:11:55	537		-	An error occurred	Logon Failure
2008-11-16 17:11:57	537		-	An error occurred	Logon Failure
2008-11-16 18:53:0	537	CT.LOCAL	-	An error occurred	Logon Failure
2008-11-16 18:53:0	537	CT.LOCAL	-	An error occurred	Logon Failure
2008-11-16 20:30:53	537		-	An error occurred	Logon Failure
2008-11-16 20:30:55	537		-	An error occurred	Logon Failure
2008-11-16 20:30:56	537		-	An error occurred	Logon Failure

Fig. 4.21 Ejemplo de un reporte en formato PDF

Para que el proceso sea automático se ha implementado la generación del reporte mediante funciones Java para que sea el propio Tomcat el que genere el documento final.

```
ResultSet dataB = pstmt.executeQuery();

ds = new JRResultSetDataSource(dataB);

// omplim report (funcio sincronitzada per evitar concurrencia de
// classpaths)

JasperPrint print = ompleReport(fCompiledReport, null, ds);
```

Fig. 4.22 Función que crea el objeto JasperPrint

Para generar el documento jprint representado como el objeto JasperPrint que muestra la figura 4.22, hemos de pasarle a la función 3 parámetros:

- fCompiledReport que es la clase que representa la plantilla jasper previamente creada con iReport.
- un HashMap con todo los “parameters”. En nuestra implementación es NULL.
- JRResultSetDatSource que representa la fuente de información que compondrá el cuerpo del reporte.

Podemos observar como mediante el ResultSet de la consulta generamos el objeto ds (JRResultSetDataSource) y se lo pasamos a la función “ompleReport” que no es más que la sentencia de a figura 4.23.

```
print = JasperFillManager.fillReport(reportStream, parametres, ds);
```

Fig. 4.23 Función que implementa los reportes.

Una vez generado el objeto JasperPrint, mediante la clase exportadora adecuada (JRXmlExporter, JRPdfExporter, etc) generamos el "array" de bytes que compone nuestro documento final. El array de bytes es volcado en un objeto tipo File para poder tener el reporte almacenado en disco.

```
exporter.setParameter(JRExporterParameter.JASPER_PRINT, print);

exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
        byteArrayOutputStream);

exporter.exportReport();

// Preparem stream de sortida

outputStream = new FileOutputStream(new File(sResultFile));

outputStream.write(byteArrayOutputStream.toByteArray());

outputStream.flush();

outputStream.close();
```

Fig. 4.24 Código para la transformación de objeto jprint a documento PDF

Para que Tomcat pueda utilizar el código de forma automática, solo hay que pasarle la ruta de las plantillas e indicarle donde guardar el documento PDF final. Para saber que plantilla y tabla utilizar se usan variables de sesión JSP que almacenan identificadores, como veremos en el capítulo 5.

CAPÍTULO 5. ESTRUCTURA DE LA INTERFAZ GRÁFICA

La interfaz gráfica, a partir de ahora GUI (graphical user interface) posibilita la interacción entre el usuario y la aplicación y muestra los resultados obtenidos.

En los sistemas de gestión de logs se suele llamar “cuadro de mando” ya que centraliza todas las operaciones posibles que puede realizar el usuario contra el sistema.

La GUI esta basada en el protocolo HTTP, es decir, es una interfaz web que podemos consultar a través de un navegador. Esto hace posible obtener un “cuadro de mando” disponible desde cualquier punto de la red de la empresa, siempre y cuando el usuario posea las credenciales adecuadas.

Como ya hemos mencionado, la GUI web esta implementada bajo el contenedor Tomcat utilizando la tecnología JSP. De manera muy genérica podemos describir esta tecnología como un híbrido de HTTP y Java, permitiendo ejecutar código programado utilizando una página web.

La GUI desarrollada en este proyecto no solamente tiene las funciones especificadas anteriormente sino que el “cuadro de mandos” nos permite el lanzamiento de procesos. El usuario escoge los parámetros de entrada mediante un formulario y la aplicación ejecuta las funciones adecuadas a éstos. A parte de JSP también se utiliza JavaScript, un lenguaje comúnmente usado en las páginas web para ejecutar código en el navegador del cliente y que nosotros utilizamos únicamente para corroborar que los parámetros de entrada de los formularios tienen valor adecuado dentro de un rango.

Tomcat tiene una estructura simple basada en directorios y es imprescindible conocerla para poder entender como funciona la plataforma ya que, como hemos mencionado, es el ejecutor de los procesos.

La jerarquía de Tomcat es la que muestra la figura 5.1 y solamente explicaremos las partes fundamentales para entender su papel de ejecutor, el resto de configuraciones pueden ser consultadas en el apartado de Anexos.

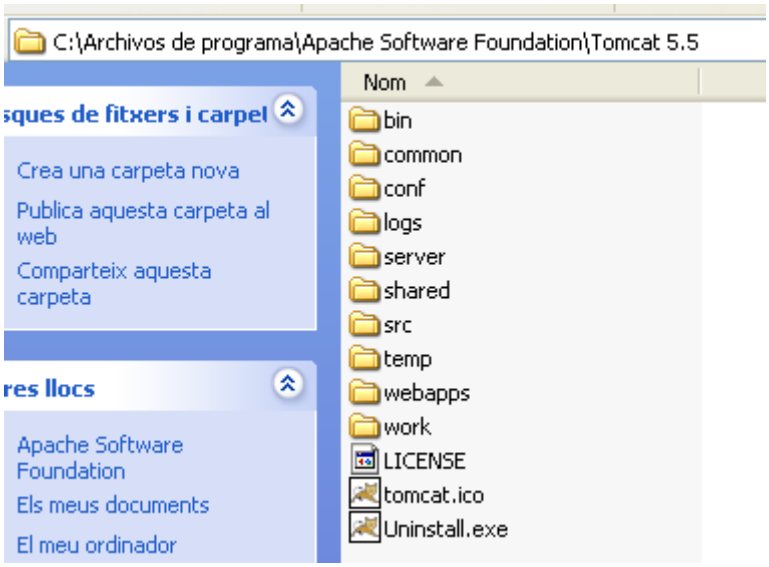

Fig. 5.1 Estructura ficheros Tomcat

A continuación explicaremos la estructura definida en TomCat:

- bin contiene los scripts de arranque y parada del servicio. Se ha de especificar la ruta de la JVM en el fichero catalina.sh para que pueda utilizarla.
- conf contiene ficheros de configuración, como el puerto de escucha, gestión de usuarios, parámetros iniciales, etc.
- webapps es la carpeta raíz donde colocaremos toda la estructura web.

Cada carpeta dentro de webapps representa una página web diferente (llamado contenedor por el léxico Tomcat) que debe de tener una estructura definida para que Tomcat la reconozca como tal. La raíz de la carpeta ha de contener un archivo de bienvenida que puede ser JSP, HTML o incluso TXT. Ha de contener una carpeta llamada “WEB-INF” que contendrá el archivo de configuración web.xml (donde se especifica entre otras cosas cual es la página de bienvenida), y la carpeta lib donde se sitúan todas las librerías que usará la aplicación. Aquí encontramos todo el código programado en Java que utilizará Tomcat como ejecutor. Las clases son comprimidas en un formato especial llamado JAR y así reducir el número de ficheros colgados de la raíz y de esta forma clarificar la estructura de ficheros.

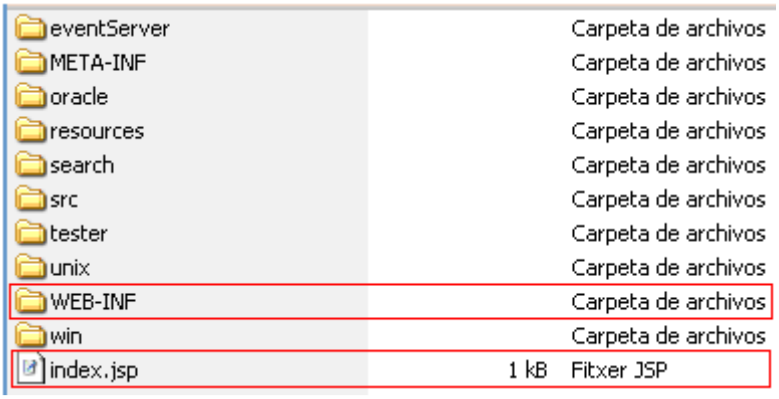

Fig. 5.2 Carpetas del contenedor de la GUI

La carpeta META-INF contiene el archivo context.xml para especificar los contextos externos de los que hace uso nuestra aplicación, en nuestro caso el contexto JDBC que hemos comentado en el capítulo 4.

Tomcat que tiene soporte para ejecutar páginas JSP, cada vez que ha de generar una página, localiza las zonas del documento donde hay código Java y la recorre de forma lineal, dejando intacto todo el código HTML. Cuando lee sentencias Java las ejecuta en la JVM pudiendo así lanzar cualquier tipo de proceso o función. Las variables Java pueden ser mostradas en código HTML si así se ha programado y de esta forma rellenar el documento final con los parámetros obtenidos. El resultado es una página HTML capaz de ser interpretada por cualquier navegador.
Una funcionalidad básica de JSP es poder generar páginas dinámicas que puedan cambiar según la interactuación con el usuario, pero en nuestro caso recogemos esas ordenes para localizar la función en la cual esta interesado el usuario y procesarla en la JVM con los datos proporcionados, si es que los hay, como muestra la figura 5.3.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page import="com.linecom.parserRequest" %>
<%@ page import="java.util.HashMap" %>
<%@ page import="com.search.*" %>
<html>                                                          Código HTML
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Throwing Seach Engine</title>
</head>
<%
SearchThread searchTh;
parserRequest parser = new parserRequest();
HashMap<String,String> myMap = new HashMap<String,String>();
//parseo los resultados del formulario
parser.parseRequest(request,myMap);   Ejecución de función Java
```

Fig. 5.3 Cabecera de página JSP

Y de esta forma se puede lanzar cualquier tipo de proceso, ya sea thread o principal, solo hemos de importar la clase e instanciar el objeto en la página JSP. Si el proceso es un thread podemos liberar la generación de la página resultante manteniendo el procesado Java en la JVM, si es un objeto principal el motor esperará a generar la página una vez acabados todos los cálculos.

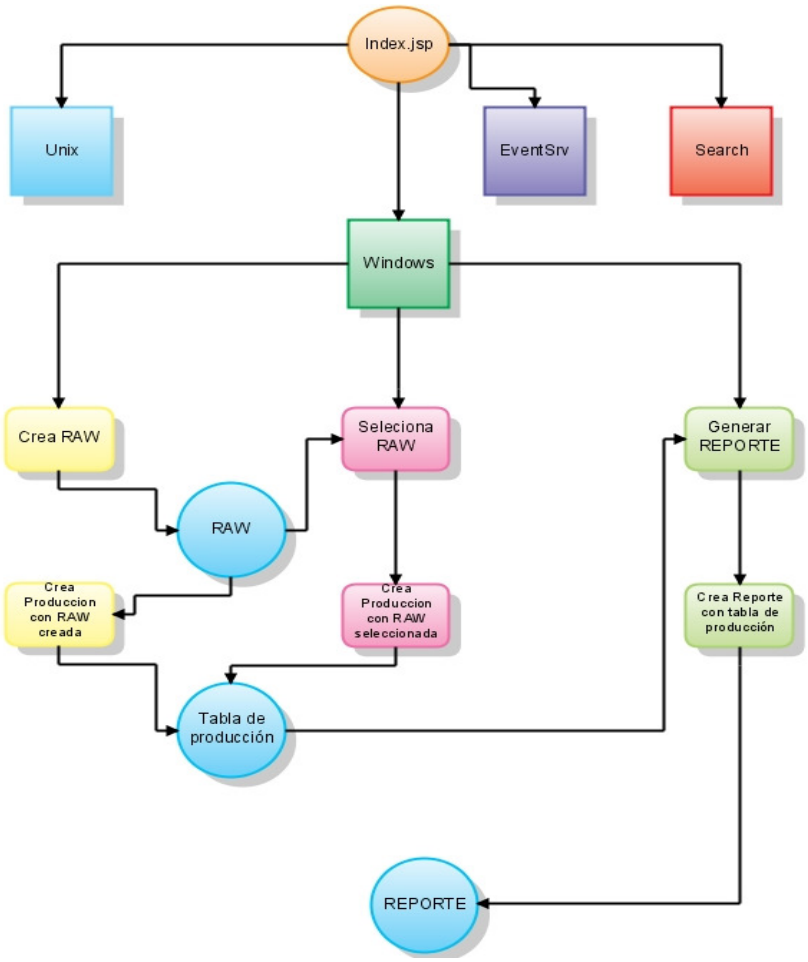

Fig. 5.4 Mapa web desplegado por la sección Windows

En la figura 5.4 vemos el mapa web que fundamentalmente es una sucesión de formularios y páginas contenedoras de resultados de operaciones, separando los distintos puntos del proceso de generación de reportes.

La estructura web contiene 4 apartados visibles desde el índice o página de presentación que son: Unix, Windows, EventSrv y Search. Las categorías de los SO son para poder trabajar de forma separada con sus respectivos logs, la de EventSrv proporciona al usuario un control sobre el servidor de recepción de eventos y Search da acceso a la plataforma de búsqueda demandada por el cliente.
Las secciones Unix y Windows son exactamente igual, solo han sido separadas para dotar de más claridad a la aplicación. El proceso queda divido en 3 partes como muestra la figura 5.5.

Fig. 5.5 Página principal de la sección Windows

En “New RAW” accedemos al formulario donde se solicita los datos necesarios para crear una tabla RAW. Como parámetros se requiere el formato del log y la ruta donde se encuentra el archivo fuente o log utilizando un cuadro de dialogo de selección de fichero, mostrado en la figura 5.6. Se requiere el formato debido a que EL CLIENTE posee sus propios agentes que extraen los logs de los sistemas y al ser distintos a los usados por nuestro sistema contienen un formato diferente a la hora de separar el contenido (caracteres separadores). Se decide cambiar de agente porque éstos han sido descatalogados y la empresa que daba soporte ha desaparecido, siendo software propietario y de código cerrado.

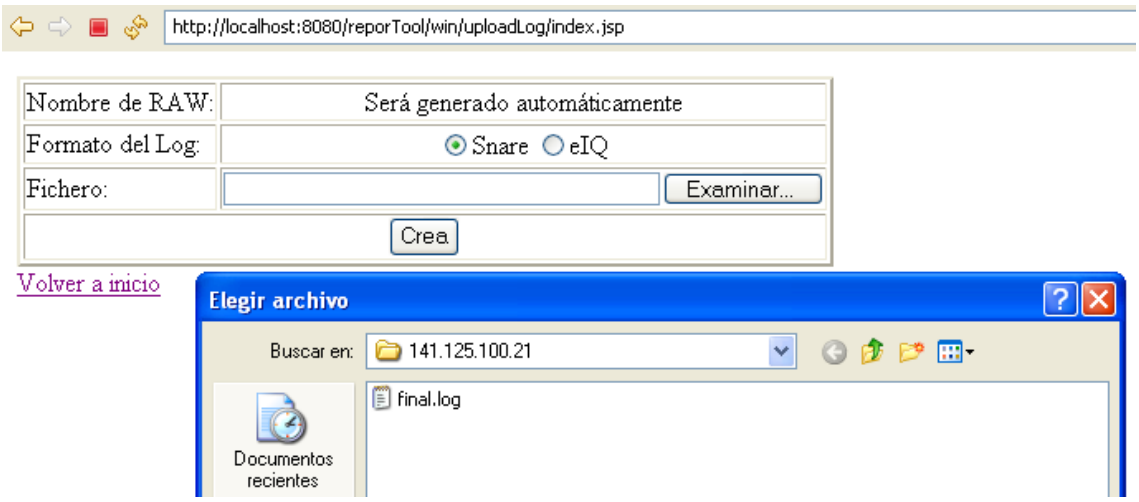

Fig. 5.6 Cuadro de selección de ruta para crear RAW

Si se produce algún error el navegador sería redirigido a una página que muestra la información de éste y el posible causante. Esto se consigue implementado una función de redirección por JavaScript e introduciéndola en los bloques try-catch que recogen cualquier excepción.
Si no se ha producido ningún error podemos continuar con el proceso pudiendo generar la tabla de producción o concatenando los resultados a una tabla ya

generada. Se da la opción de concatenar para poder generar logs que contengan más de un log de información.
La página de creación de tabla de producción es otro formulario, como se puede ver en la figura 5.7.

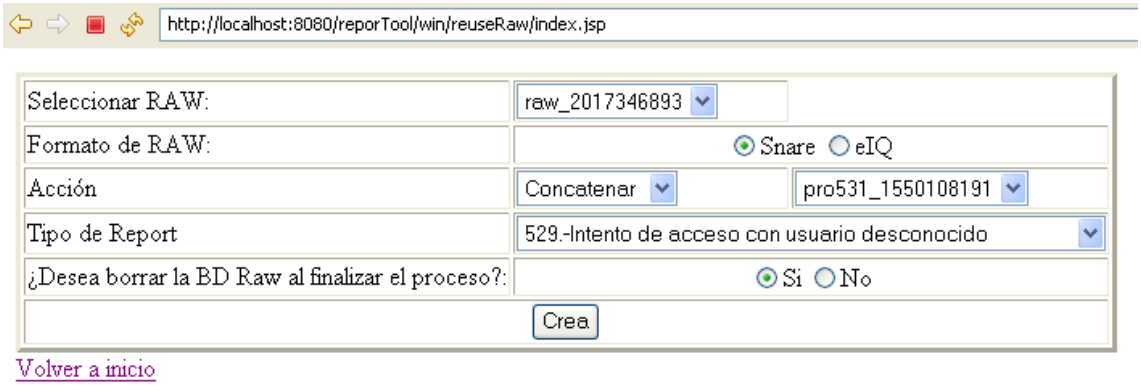

Fig. 5.7 Formulario de tabla de producción

En el primer selector se cargan los nombres de todas las tablas RAW de formato Windows que existen en la BD. En la línea de “Acción” podemos elegir concatenar, con lo que podremos acceder a otro selector con los nombres de las tablas de producción cargados, o crear una nueva tabla.
Una vez obtenida la tabla, el tercer paso es generar el reporte. Una vez más, tenemos un nuevo formulario con un selector cargado con los nombres de las tablas de producción y otro con los diferentes tipos de reportes. Al finalizar el proceso podemos ver desde nuestro navegador el documento PDF final. Esto se consigue leyendo el fichero de forma binaria y enviando los resultados a la función “response” que representa el canal donde se envían los datos HTML al navegador.

En el apartado de configuración del servidor de captura de eventos podemos configurar la tabla de conversión del servicio DNS, pudiendo visualizarla desde el navegador para poder observar las modificaciones. En esta sección han quedado pendientes varias tareas a implementar que por falta de tiempo y por tener poca prioridad no se han realizado. La primera es poder controlar el servicio del servidor desde la interfaz web, pudiendo iniciarlo, pararlo o reiniciarlo. La segunda tarea es poder configurar los eventos auditados desde la GUI también, en este sentido para Windows solo habría que generar una lista con las máquinas auditadas y mostrar un link al panel de configuración del agente que también es web, para Unix habría que modificar de forma remota el fichero de configuración de Syslog.

La sección de búsqueda será comentada conjuntamente con el código en el capitulo 6.

CAPÍTULO 6. AUTOMATIZACIÓN Y BÚSQUEDA

6.1. Automatización

Un requisito del cliente es que el sistema actuase de forma automática, es decir, que fuera capaz de almacenar los logs en tablas, parsear la información y generar las tablas de producción de una forma transparente al usuario, de manera autónoma y con una cierta periodicidad.
Para ello se utilizan hilos de ejecución que no tengan la necesidad de ningún tipo de parámetro de entrada y así evitar la intervención del usuario durante el proceso.
El primer parámetro que se ha de resolver es bajo que periodicidad los logs serán tratados. El cliente reclamó el requisito de separar la información de eventos de forma mensual, pero después de realizar pruebas de rendimiento del sistema sobre logs mensuales, los niveles de utilización de la CPU superaban el 80% durante prácticamente todo el proceso y no solo eso, cualquier fallo en la estructura del log generaba una cancelación de la transacción de la BD, es decir, eliminar cualquier modificación para restaurar los datos originales, en nuestro caso volver a dejar la tabla RAW en blanco. Es por estas razones que se decidió procesar los registros de forma diaria. De esta manera solucionamos los problemas que puede aparecer por un fallo del sistema y evitamos también los elevados niveles de ejecución que se mostraban en el primer caso.
A continuación detallamos los procesos que deberá seguir la plataforma de manera diaria:

- Diariamente un proceso “congelado” despertará, preferentemente por la noche, a partir de las 00:00 y revisará la ruta donde se almacenan los logs.
- Se identificarán los logs pertenecientes al día anterior y se seleccionaran para su procesado.
- Por cada log se generará una tabla RAW con el contenido de éste.
- Por cada RAW se extraerán y generaran tablas de producción. Si es principio de mes las tablas de producción serán creadas antes de iniciar el proceso de extracción, sino los resultados serán concatenado a la tabla correspondiente.

Con el fin de no crear excesivas tablas en la BD, también se ha creado un proceso mensual que borre todas aquellas tablas de producción que no contengan ningún elemento.

La automatización, como hemos mencionado, empieza con un proceso en segundo plano llamado TaskManager.java, siendo su única función comprobar la hora en la cual nos encontramos y si es la adecuada, lo más conveniente es una hora nocturna, lanzar el proceso de tratamiento de datos, Figura 6.1.

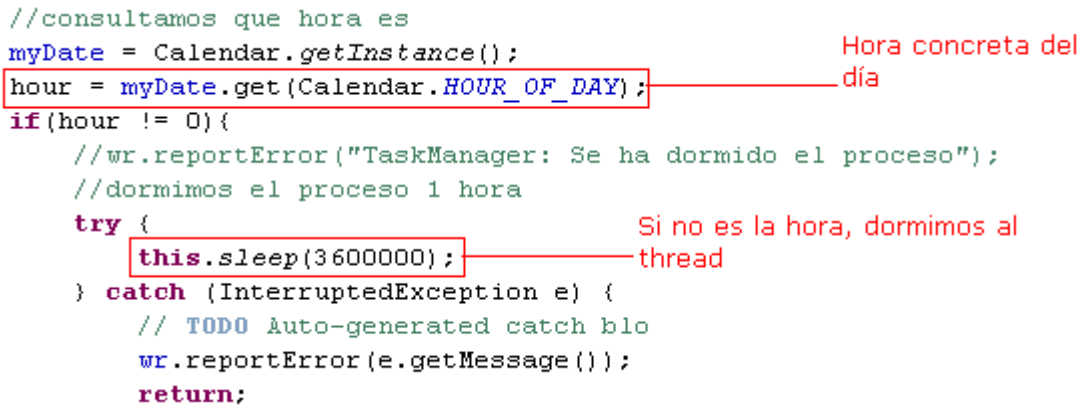

Fig. 6.1 Lanzador del proceso diario

El proceso se podría haber implementado bajo un administrador de tareas pero eso incluye una implementación distinta para cada sistema operativo.

Lo primero que realiza el proceso de tratamiento de datos es analizar el directorio donde se guardan los logs. La ruta está definida en una tabla llamada “settings”. Mediante la clase Calendar el sistema puede concretar la fecha del día anterior y así compararla con la de los logs. Gracias al formato con el que guarda nuestro EventServer los logs, simplemente leyendo el nombre del archivo se pueden extraer la máquina a la que pertenecen, sistema operativo y el día de registro.

A continuación el proceso se divide en varios hilos de ejecución por cada log del día anterior para dotar de rapidez al proceso. En el diagrama de la figura 47 observamos un control de threads, que es una clase común que cuenta el número de hilos en ejecución y no permite nuevas ejecuciones si se ha rebasado el limite establecido. Este mecanismo de seguridad se ha implementado para no saturar la máquina ni la memoria de la JVM.

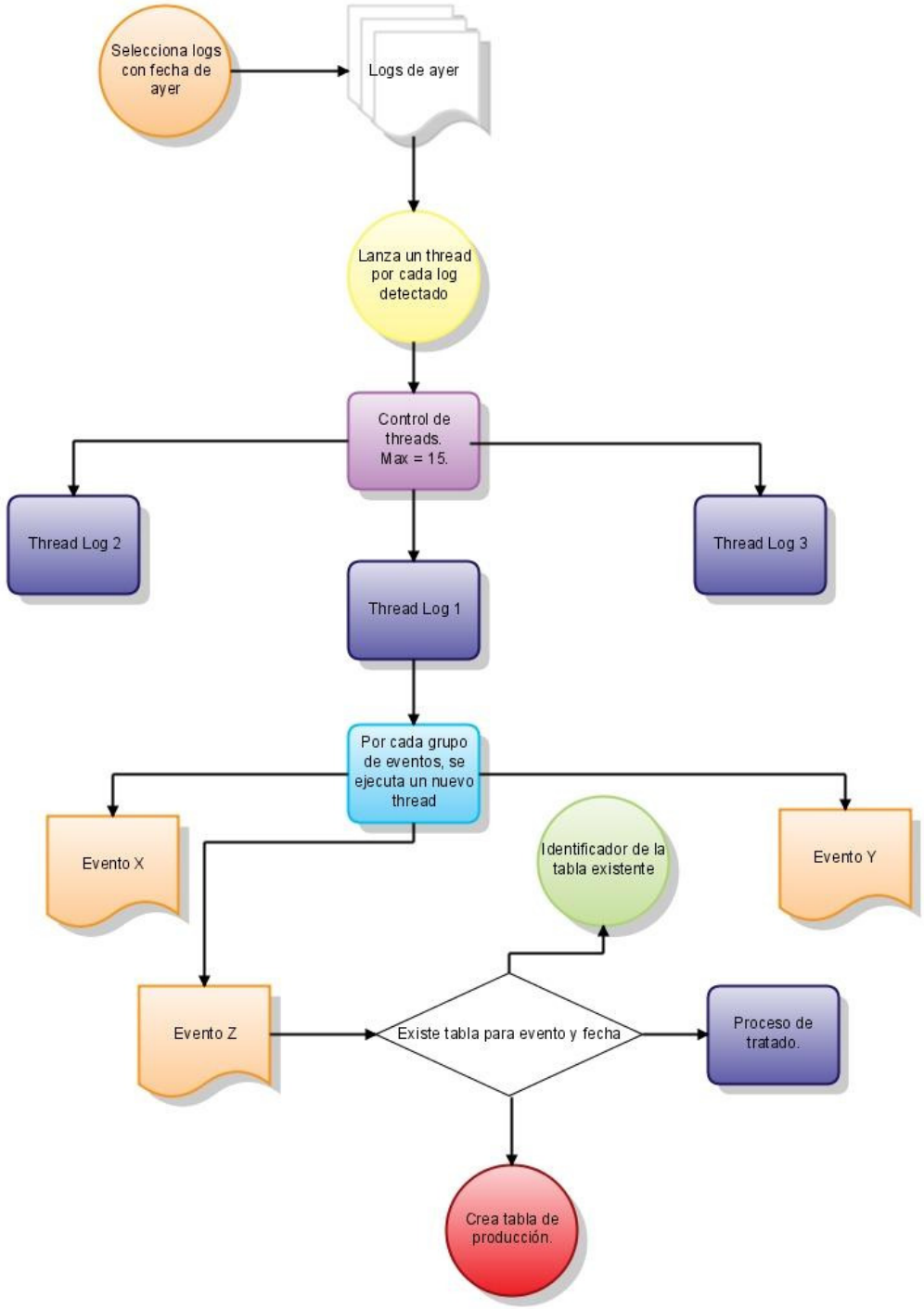

Fig. 6.2 Proceso de automatización

Por cada hilo de ejecución y una vez identificado el sistema operativo, de la misma forma que se identifica la fecha, la aplicación puede generar una tabla a medida del log y de esta forma volcar los datos contenidos en él.

Una vez creada la RAW el proceso se divide de nuevo para otorgar un nuevo hilo de ejecución a cada proceso de extracción de evento, es decir, por cada evento diferente que se pueda extraer de la RAW se creará un nuevo thread.
Cada nuevo hilo extrae el evento que se le ha asignado, formatea el contenido y lo almacena en una tabla específica para ese tipo de eventos.
Ya hemos mencionado que existen una serie de eventos que se almacenan en una misma tabla por pertenecer al mismo tipo, por lo tanto varios hilos de ejecución leen y escriben sobre los mismos objetivos, siendo esto posible por la propiedad transaccional de MySQL a la hora de escribir evitando que lo hagan a la vez y la propiedad multi-usuario que permite que varios procesos puedan leer sobre una misma tabla.
Para finalizar el proceso y completar el ciclo se deben de borrar las tablas RAW y los logs. Para ello se crea una nueva clase común de registro de errores y que es implementada en cada hilo de producción y es examinada en el momento en el que todos los hilos han finalizado su proceso y se procede a borrar la tabla RAW. Como mecanismo de seguridad para evitar la pérdida de datos, si existe algún error en la clase auxiliar, no se borrará el log original como podemos ver en la figura 6.3.

```
//si no se ha producido ningun error, borramos el log
if(errorBuffer.isEmpty()){
    File deleteLog = new File(pathLog);
    deleteLog.delete();
}
wr.reportError("Proceso de Log finalizado...");
```

Fig 6.3 Mecanismo de control al borrar logs

Todos los hilos tienen otra clase común, *wr* en la figura 6.3, que tiene como objetivo reportar cualquier acontecimiento destacable durante el proceso hacia un archivo de la máquina local. La ruta de logs de la aplicación viene definida en la tabla “settings”.

6.2. Sistema de búsqueda

El sistema de búsqueda es un requisito demandado por el cliente implementado en el “panel de control” como hemos podido ver en capítulo 5. El motor se basa en la información contenida en la base de datos.
Para poder utilizar la base de datos de manera eficiente es indispensable ordenar e indexar todas las tablas de producción, con lo que se crea, para este fin, una tabla-índice que contiene como parámetros de columna: el identificador de la tabla de producción, la IP o nombre de la máquina a la que pertenecen los datos, los eventos y el periodo temporal contenido. La tabla-índice es consultada cada vez que se busca un evento por identificador, máquina y/o periodo temporal. Esta tabla también es utilizada por el proceso de automatización para averiguar si una tabla de producción para una máquina y fecha determinadas existe o no.

Una vez ordenada la información, se puede montar el proceso de búsqueda encima. El proceso se divide en 2 fases:
En la primera fase el usuario podrá localizar eventos por el nombre de éstos (identificador) y el segmento temporal en el cual se han producido. Después que el motor realice la criba de resultados, el usuario, podrá iniciar la segunda fase si lo desea. Esta segunda fase permitirá filtrar el resultado obtenido utilizando como variable de entrada los parámetros propios del evento. Por ejemplo, si nos centramos en un evento Windows 644 (bloqueo de cuenta), el usuario podrá filtrar en esta segunda fase todos aquellos eventos que sean referidos a una cuenta, dominio, máquina concreta.
La división del proceso en 2 fases viene dada por el hecho que hasta que el usuario no ha seleccionado el evento a buscar no disponemos de información necesaria para poder preparar el sistema de filtrado a través de los parámetros propios del evento.

Si nos centramos en el funcionamiento interno del motor de búsqueda, podemos apreciar que actúa directamente sobre la información contenida en la BD, en base a la información proporcionada por el usuario y con la finalidad de crear una tabla que contenga todos los eventos buscados por el usuario. El seguimiento del proceso se puede observar en la figura 6.4.

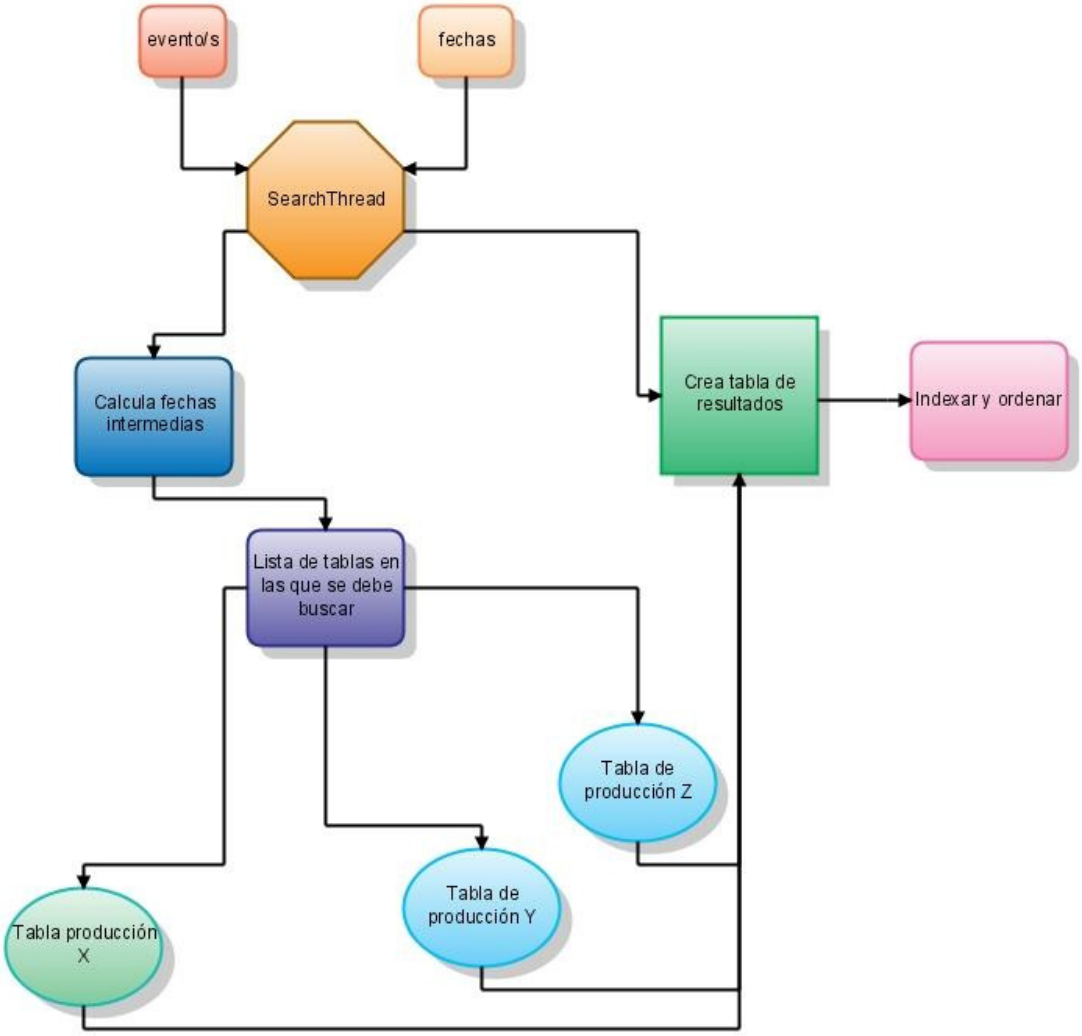

Fig. 6.4 Diagrama del proceso de búsqueda

En la primera fase al motor se le pasan los parámetros evento, fecha inicial y fecha final (mediante el formulario que mostramos en la figura 6.5).

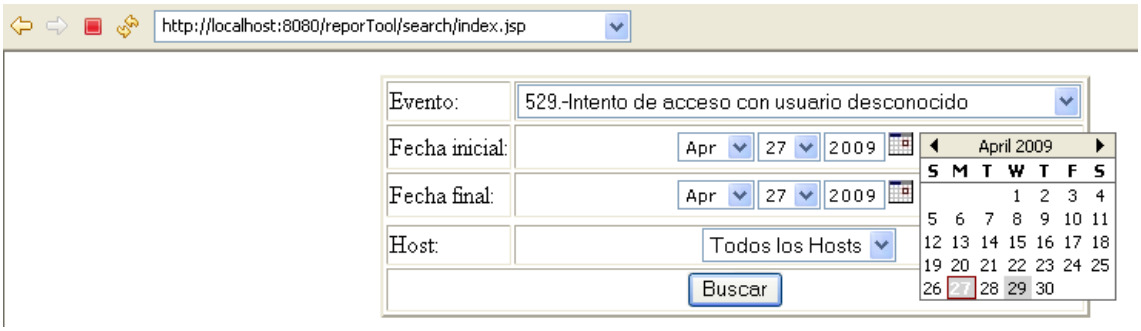

Fig. 6.5 Interfaz de búsqueda

Con los datos de entrada se inicia del proceso principal de búsqueda que tiene como objetivo crear una tabla con todos los resultados que encajen con el patrón proporcionado. La lógica de la ejecución es la siguiente:

- Se crea una tabla vacía con la estructura del evento buscado.
- Ya que el usuario ha proporcionado una fecha inicial y otra final se obtienen los rangos de fechas intermedias mensuales. Es decir si el usuario introduce como fecha inicial 12/03/2009 y final 15/05/2009, la aplicación genera 3 rangos de búsqueda: 03-2009, 04-2009 y 05-2009. Cada rango es un objeto HashMap. Para poder buscar específicamente los días el primer y ultimo mes de búsqueda llevaran una clave "day".
- Por cada objeto resultante del paso anterior se recupera mediante la tabla-índice la tabla que contiene dicho evento para ese mes concreto y se almacena en el objeto el nombre.
- Los objetos HashMap que contienen fecha y nombre de tabla son almacenados en un vector para ser leído mediante un bucle y lanzar por cada uno un thread que extrae el contenido coincidente con los parámetros de búsqueda y lo inserta en la tabla que se creó en el primer paso, llamada tabla-resultado.
- Se implementa una clase controladora de threads para permitir al proceso principal "saber" que todos los hilos lanzados se han ejecutado correctamente para poder continuar el proceso.

El paso de rango por fechas mensuales se ha implementado de esa forma para agilizar las búsquedas grandes, ya que si los periodos temporales contienen meses enteros, el mecanismo de búsqueda no ha de comparar el campo DataTime de cada fila de la tabla con el proporcionado por el usuario, agilizando enormemente el proceso. Han sido elegidos periodos mensuales porque cualquier otro más pequeño generaba muchas tablas vacías por falta de contenido e incrementaba el número de tablas-objetivo en las cuales buscar.

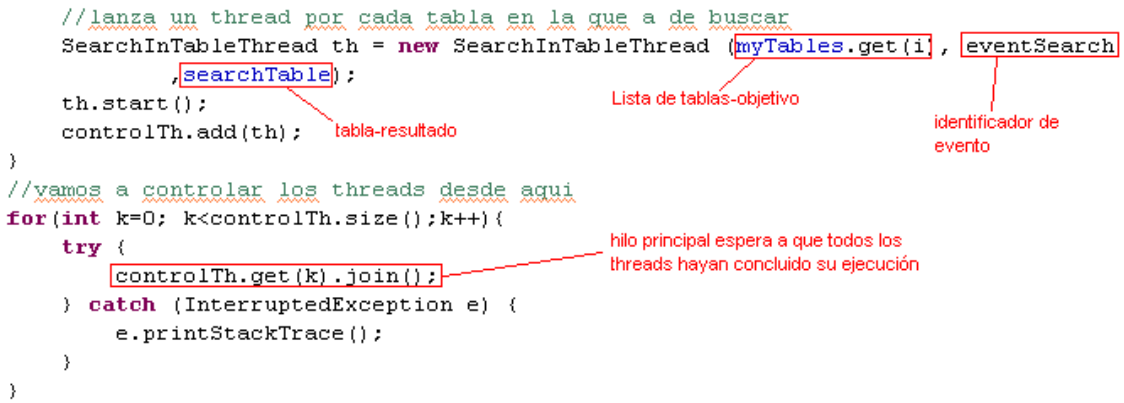

Fig 6.6 Proceso multi-hilo para la búsqueda y control de threads

Una vez obtenida la tabla-resultado, se creará un índice MySQL en el campo "DataTime" para agilizar su posterior ordenación por fecha. La tabla de resultados es copiada a una nueva tabla que contiene un identificador numérico por fila para poder paginar los resultados. Mediante esta nueva tabla se muestran los resultados al usuario de forma paginada, cada vez que el usuario pide mostrar más resultados, el motor genera una consulta a la tabla-resultado.

En la segunda fase, se trabaja con la tabla-resultado anterior y con parámetros de sesión HTTP. En ella podemos hacer un filtrado más profundo del resultado obtenido, pudiendo hacerlo por parámetros propios del evento. Cuando el usuario ha elegido el evento y las fechas, éstos son guardados en una sesión para su uso posterior. Al presentar los resultados se carga un formulario de filtrado, como vemos en la figura 6.7, que contiene parámetros específicos del evento, siendo esto posible gracias a que la aplicación reconoce el evento a través de los parámetros guardados en sesión.

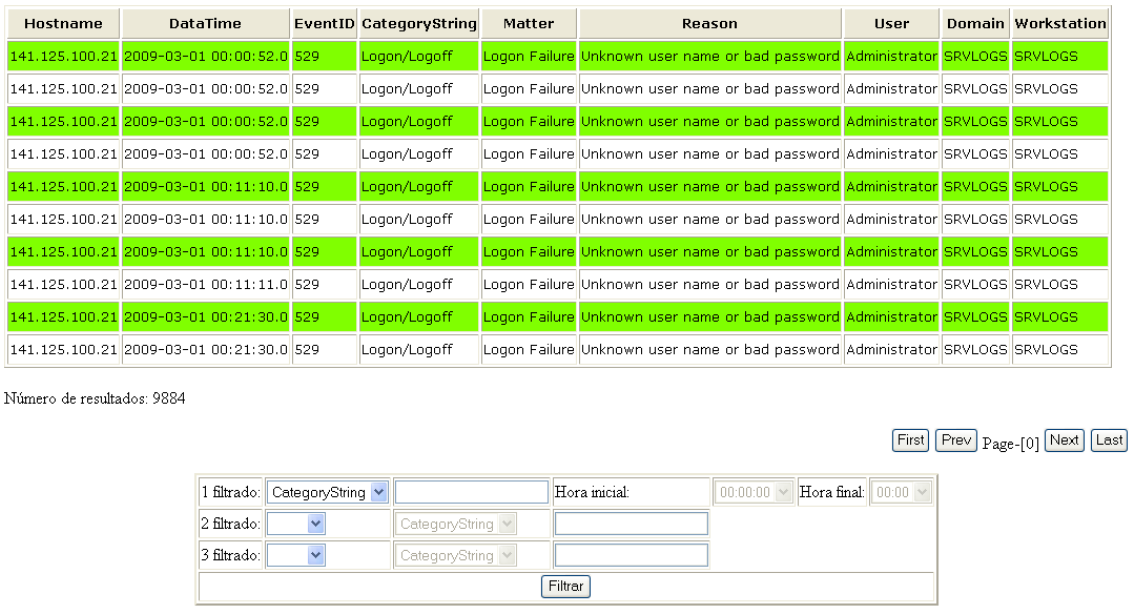

Fig. 6.7 Interfaz de resultados de búsqueda

El sistema de filtrado propio de evento trabaja con una única tabla-objetivo (tabla-resultado de la fase 1), se recoge una lista clave-valor con los parámetros de filtrado, extraídos del formulario, y mediante sentencias SQL se extraen los eventos coincidentes, guardándolos en una nueva tabla de resultado. Se realiza un nuevo almacenamiento para poder realizar varios filtrados recursivos del contenido y para poder paginarlo sin saturar la memoria. Existe la opción de poder filtrar por un intervalo de horas si la fecha inicial coincide con la final, es decir, si la búsqueda se realiza bajo el mismo día, mes y hora. Esta opción no se implementó en la búsqueda principal por problemas de rendimiento, ya que se ha de realizar una comparación por cada registro de cada tabla del campo “DataTime” y en búsquedas de varios meses significaba una espera excesivamente larga para el cliente.

Fig. 6.8 Filtrado por horas

Se ha implementado la posibilidad de exportar los resultados utilizando el mismo mecanismo que para reportar los eventos, con las librerías JasperReports y las plantillas ya confeccionadas. Al ser las tablas-resultado del mismo formato que las de producción, no se ha modificado nada del módulo de reporte para poder implementar en el motor de búsqueda.

CAPÍTULO 7. SISTEMAS DE MEJORA Y RENDIMIENTO

En este capítulo se quiere reflejar la problemática de trabajar con gran cantidad de datos y las implicaciones a nivel de código que ello provoca en la plataforma.

Para poder mostrar estas implicaciones, el siguiente capítulo lo dividiremos en dos puntos diferenciados. Por un lado nos centraremos en dos problemáticas aparecidas durante el desarrollo de la aplicación y la solución presentada. Por el otro lado, nos centraremos en parámetros de rendimiento observando diferentes procesos realizados en la plataforma..

7.1 Sistema de mejora

En primer lugar, el origen del volumen de datos viene condicionado por EL CLIENTE, no solo por poseer una red de grandes dimensiones sino porque en una fase inicial reclamó los reportes de eventos directamente desde sus propios agentes, como ya hemos comentado anteriormente, no realizándose ninguna restricción sobre los eventos a reportar dando lugar a unos logs de medidas innecesariamente grandes.

Al trabajar con grandes cantidades de datos surgieron 2 problemas concretos:

- Debido al incremento de tiempo en el procesado de la información algunas aplicaciones externas de la plataforma generaban un “timeout” por la larga espera que se traducía en un corte de la conexión.
- Al incrementarse la carga computacional, se incrementaba la necesidad de memoria, la cual es limitada por hardware y más concretamente por los parámetros de configuración de la JVM.

A continuación especificaremos de forma más detallada estos 2 problemas, así como las implementaciones realizadas para agilizar el comportamiento de la plataforma.

7.1.1 Timouts del navegador

El primer problema lo encontramos en el “cuadro de mandos” en referencia al navegador web. Cuando un usuario envía un formulario para ejecutar un proceso, el navegador envía una petición al servidor que es contestada cuando el proceso ha terminado. En ese espacio de tiempo el navegador entra en estado de espera, pudiendo cerrar la conexión si dura excesivamente, parando de esta manera la ejecución de los procesos en la máquina virtual. Este comportamiento corresponde a un mecanismo de seguridad que evita un posible desbordamiento de memoria, ya que si las conexiones siguen activas, los recursos necesarios para mantenerlas no son liberados y una acumulación de conexiones puede dar lugar a una falta de memoria. El tiempo de espera depende del navegador, siendo de unos 900 segundos para Internet Explorer, que en nuestro caso era insuficiente para el procesado de algunos logs mensuales.
El procesado, como ya hemos visto a lo largo del documento, consta de 2 fases críticas: creación de RAWs y generación de tablas de producción. Son 2

procesos distintos, es decir, 2 peticiones distintas del navegador. Los tiempos de computación son directamente proporcionales a la cantidad de datos de entrada y en nuestro entorno las cotas más altas se alcanzan para las máquinas controladoras de dominio, pudiendo alcanzar un total de 30 millones de registros para una RAW con un tiempo aproximado de ejecución de 3000 segundos y una extracción de 7 millones de eventos a una tabla de producción mensual, consumiendo un tiempo de unos 7500 segundos. Esta franja de tiempos no es igual para todas las audiciones ya que existen logs de máquinas que no contienen más de 5 Mb de información, en cuyo caso el tiempo de ejecución es inferior al segundo, pero necesitamos adecuar el rendimiento de la plataforma precisamente para los casos extremos para de esta manera abastecer de servicio a todos los logs, sea cual sea su tamaño.

Una vez analizado el problema, se considera la posibilidad de liberar el proceso principal de la JVM. Este paso implica la creación de una página de espera intermedia que controle el thread de procesamiento de datos en ejecución. Dicha página se recarga automáticamente cada cierto tiempo para actualizar la información del proceso, de esta forma la petición es resuelta y el navegador no detiene la ejecución. Por el contrario, realizamos un mayor número de peticiones al servidor web.

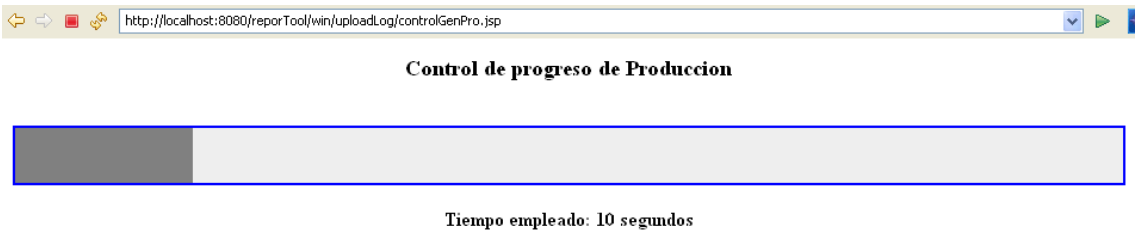

Fig. 7.1 Página de espera

Para poder liberar el proceso principal, la ejecución se lanzará desde un hilo secundario. Debemos hacer hincapié en el control de dicho hilo ya que al no tratarse del hilo principal el navegador no podrá supervisarlo y por lo tanto nos quedamos sin un canal donde poder pararlo o vigilarlo. Para ello controlamos 2 parámetros, el primero es el propio thread, entendido como un objeto Java y el segundo es el identificador de la tabla resultante de proceso, ya que es totalmente aleatorio (el identificador se obtiene mediante una función que genera números aleatorios) y al no disponer de canal de salida de parámetros hemos de conocerlo previamente. El control se realiza mediante el JavaBean session implementado por el servidor Tomcat. Session lo que permite es guardar cualquier objeto Java en una variable general capaz de ser utilizada en todas las páginas de la aplicación web, realizándose el proceso a través de galletas para mantener separadas las variables de varios usuarios. Así pues, guardamos el identificador integer y el objeto thread en session y por cada refresco de la página se recupera el objeto thread y se interroga para saber si ha finalizado o no, como muestra la figura 7.2.

```
//recogemos la clase controladora de proceso
Thread winraw = (Thread)session.getAttribute("winpro");

<% if(!winraw.isAlive()) { %>
<script type="text/javascript">
    var segundos = 5;

    function redireccion() {
        window.location.href = 'genReport.jsp';
    }

    setTimeout("redireccion()",segundos);
```

Fig. 7.2 Control de proceso secundario mediante bean session.

Si el proceso ha finalizado, redirigimos el navegador hacia la página siguiente del proceso mediante una función JavaScript.
Todo este aislamiento del proceso pesado en computación no solo evita que los navegadores puedan parar la ejecución sino que además mejora sustancialmente la eficiencia de la aplicación ya que separamos de la lógica JSP la computación, evitando recargas de código e interrupciones.
De esta manera se ha implementado en todas las tareas que requieran un computo elevado como la creación de RAWs, tablas de producción para los formatos Snare&Syslog y eIQ y creación de reportes.

7.1.2 Consumo de memoria

Toda variable que no sea almacenada en el disco duro durante un proceso, se almacena en memoria. Por lo tanto, cuando ejecutamos un proceso que contiene gran cantidad de variables o variables de gran tamaño, el espacio requerido es mayor. Si las variables del proceso provienen del contenido de un log de 5 GB probablemente la máquina se quede sin memoria RAM utilizable.

La JVM se ejecuta como un proceso independiente al cual se le especifican los parámetros de de memoria mínimos y máximos que puede utilizar para no colapsar la máquina en la cual se ejecuta. Si un proceso de la JVM requiere más cantidad de memoria de la que posee en ese momento, se detendrá la ejecución y se generará una excepción denominada “Java Heap Space”.
El problema de memoria resulta de la carga de información de la RAW, es decir al extraer los eventos auditados. Debido a que las filas de la RAW no están identificadas numéricamente no podemos realizar una carga progresiva de los resultados. Las librerías JR por su implementación necesitan como parámetro de entrada un objeto ResultSet con todos los datos que se van a reportar en el documento, Figura 7.4. Debido al funcionamiento de dicho objeto que almacena su contenido directamente en memoria, la aplicación detenía su ejecución, imposibilitando realizar tablas de producción y reportes de grandes logs.
Siendo el objeto ResultSet un puntero a memoria existe una configuración utilizando el conector JDBC para cargar una a una las filas de la consulta. Esta se realiza como muestra la figura 7.3 mediante el objeto *PreparedStatment* que

se encarga de establecer el canal de comunicación con MySQL. Para realizar la conexión le indicamos los parámetros TYPE_FORWARD_ONLY y CONCUR_READ_ONLY que establecen que no se podrán leer filas por detrás del puntero y que no se puede realizar ningún tipo de modificación en el objeto ResultSet, combinado con la función setFetchSize que indica a JDBC la cantidad de filas que ha de cargar cuando se demandan resultados, hacen que la carga sea progresiva, evitando una sobrecarga de memoria.

```
//preparamos el statment con valores para que recoja 1 a 1 las rows
PreparedStatement pstmt = conn.prepareStatement(varSQL, java.sql.ResultSet.TYPE_FORWARD_ONLY,
        java.sql.ResultSet.CONCUR_READ_ONLY);
pstmt.setFetchSize(Integer.MIN_VALUE);

pstmt.setInt(1, name);
pstmt.setInt(2, eventsVector.get(j));

ResultSet dataB = pstmt.executeQuery();
```

Fig. 7.3. Configuración para poder leer de forma secuencial la BD

El problema de la falta de espacio también repercutió al proceso de reportes, no solo por la necesidad de un ResultSet con todos los datos, sino porque al realizar el reporte, las páginas son almacenadas según se generan en memoria hasta que el documento está completo (debido a su implementación). Este modo de funcionamiento puede agotar la memoria para documentos excesivamente grandes.

```
ResultSet dataB = pstmt.executeQuery();
ds = new JRResultSetDataSource(dataB);
try {

    // // omplim report...

    print = JasperFillManager.fillReport(reportStream, parametres, ds);

} catch (JRException ex) {
    throw ex;
```

Fig. 7.4 Implementación de un reporte usando una BD

JR ya había detectado el problema y implementado un mecanismo de virtualización, que consiste en almacenar datos en archivos temporales del disco duro. Evidentemente al funcionar de esta manera el tiempo necesario para generar reportes se incrementó ya que el acceso a disco es siempre más lento que el de memoria, pero era la única solución viable para dotar a la aplicación de un mecanismo de reporte consistente y funcional. La configuración adicional para las librerías JR se muestra en la figura 7.5.

```
// creamos el "swap file" con 1024 bytes por bloque y 40 full
// bloques para que crezca el archivo
JRConcurrentSwapFile sfile = new JRConcurrentSwapFile(temporalDir
        .getCanonicalPath(), 1024, 40);
// creamos el virtualizador que usa el sfile. 40 bloques como máximo
// y posibilidad para borrar el sfile "true"
JRSwapFileVirtualizer virtualizer = new JRSwapFileVirtualizer(40,
        sfile, true);

// JRFileVirtualizer virtualizer = new JRFileVirtualizer (10,
// temporalDir.getCanonicalPath());
Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put(JRParameter.REPORT_VIRTUALIZER, virtualizer);
```

Fig. 7.5 Implementación de virtualización JR

7.2 Rendimiento de la aplicación

Al tratarse de un sistema con una gran carga de datos, al rendimiento computacional se le otorga un gran valor. Evidentemente siempre existirán limitaciones de hardware pero existen varios mecanismos e implementaciones para ciertos aspectos de nuestra plataforma que permiten mejorar sus tiempos de procesado y respuesta, liberando al sistema de una imagen de funcionamiento tosco y lento. En está sección se explican los grandes problemas de rendimiento y las soluciones aportadas.

7.2.1 Tiempos de ejecución

Debido al gran número de datos a procesar existe un incremento en el tiempo de ejecución para cada proceso. Este hecho no solo afecta a la plataforma de forma cuantitativa (timeouts, exceso de memoria, etc) sino que también lo hace de forma cualitativa ya que el usuario ha de esperar largos periodos de tiempo para obtener los resultados. Es por dicha razón que se han realizado esfuerzos en este sentido para agilizar los procesos lo máximo posible, teniendo en cuenta que siempre estaremos limitados por el hardware de la máquina que sustenta a la aplicación.

7.2.2 Indexado de tablas MySQL

En el entorno de base de datos existe la posibilidad de poder crear una relación entre la posición de filas y su contenido, llamada índice. El objetivo de esta característica es agilizar la lectura y recuperación de datos de una tabla. Por el contra tener un índice implica una deceleración en todas las opciones de escritura de la misma tabla, debido a la actualización de los datos relacionantes, es por ello que éstos suelen ser usados en tablas estáticas y reutilizables. Los índices son utilizados en campos cuyo contenido es heterogéneo, insertar un índice en un campo que solo tenga 4 valores posibles no incrementa la eficiencia de lectura y dificulta la escritura.

Por todo esto es importante escoger la tabla y el campo adecuados, que en nuestro caso son los siguientes:

- Campo EventID para las RAWs de Windows
- Campo DataTime para las tablas de producción.

En el caso de la RAW de Windows podemos indexar el EventID, aunque este parámetro se suele repetir y genera un índice pesado en disco pero incrementa significativamente el proceso de creación de tablas de producción.
El índice de DataTime no es excesivamente rápido ya que se basa en la comparación de cadenas de caracteres sin embargo se ha implementado por la necesidad de dotar de la máxima rapidez al sistema de búsqueda.

7.2.3 Ejecución multi-hilo

La ejecución multi-hilo tiene como objetivo aprovechar la función multi-proceso de los sistemas operativos implementados actualmente. Lo que se consigue en este modo de ejecución es dividir la tarea a procesar en diferentes segmentos que se ejecutan a la vez, reduciendo el tiempo de ejecución. Dicho proceso ha sido ampliamente utilizado como hemos descrito en los capítulos 3,4 y 6 para el almacenamiento de paquetes, creación de RAWs, tablas de producción y sistema de búsqueda, respectivamente. Por lo tanto en esta sección, explicaremos como generar un hilo y las posibles consecuencias en código que pueden tener.
Un hilo secundario potencial es todo objeto que extiende a la clase Thread y sobrescribe su método run(), como muestra la figura 7.6. El método run() contiene el código que se ejecutará en un hilo diferente al principal. Hemos de crear un constructor que abastezca al hilo de los parámetros de entrada y también de salida, ya que como hemos mencionado anteriormente en este capítulo, el control del thread es complicado y hemos de anticiparnos a los posibles resultados.

Implementación de una clase Thread

```
public class CreateProSetOfEvents extends Thread {

    public void run() {
        // recupero el tipo de raw que es
        String config = mMap.get("config");
```

Lanzamiento en JSP de un Thread

```
//Lanzamos el proceso
CreateProSetOfEvents winpro = new CreateProSetOfEvents(myMap,tableTarjet, buffer);
winpro.start();
session.setAttribute("winpro", winpro);
```

Fig. 7.6 Implementación de un thread

Al trabajar con threads se ha de controlar su generación y destrucción, ya que la acumulación de excesivos threads en ejecución puede colapsar la máquina por desbordamiento de recursos, ya que por cada uno de ellos siempre se reserva una parte de éstos (básicamente memoria). Para ello hemos construido una clase controladora que se basa en un mecanismo de cola finita. La clase

implementa métodos síncronos para la manipulación de los elementos de la misma, es decir que solo un thread puede introducir/extraer simultáneamente elementos. Cuando se crea un thread se elimina un testigo de la cola y cuando éste se destruye se introduce de nuevo. De esta manera cuando el proceso principal quiere ejecutar un nuevo thread ha de consultar la cola, si está vacía se duerme un periodo de tiempo finito para luego volver a consultarla, si existe un testigo en la cola, ejecutará el proceso secundario.
El control es necesario también para no colapsar la base de datos, ya que cada nuevo thread, por implementación, crea una nueva conexión con la BD y esta posee la capacidad de manejar un número limitado a la vez.

CONCLUSIONES

El incremento del número de elementos que forman parte de una red empresarial debe de ser acorde con la necesidad de seguimiento de su comportamiento.
Este aspecto, en la práctica, es muy poco valorado por infinidad de empresas y corporaciones que prefieren gastar sus presupuestos en “curar” y no en “prevenir” cualquier alteración que se produzca en sus sistemas. Evidentemente la productividad puede quedar afectada en cualquiera de ellos, pero si hablamos de los elementos que velan por la seguridad de la información corporativa, un evento a este nivel puede afectar a la imagen de la empresa sin contar las pérdidas económicas que siempre se producen.
El seguimiento del registro de actividades de los sistemas de seguridad permite descubrir comportamientos sospechosos, indicadores de un posible ataque, mejorando sustancialmente el tiempo de reacción de la empresa, evolución y verificación del rendimiento del sistema pudiendo diferenciar los puntos menos eficaces, control y cómputo de las operaciones realizadas y en caso de un ataque realizado es la única fuente de información que puede indicar como y/o quien lo ha realizado.
Es por tanto imprescindible realizar una supervisión adecuada de la información que nuestros sistemas de seguridad son capaces de generar en su ejecución. La dificultad para manejar el volumen de información generada por el conjunto de elementos, el precio de los productos corporativos específicos para facilitar esta tarea y al tratarse de un producto que directamente no aporta beneficios a la empresa, ya que realiza funciones de prevención, propician que el seguimiento de este tipo de eventos sea escueto y no requiera excesivo tiempo en muchos casos.
Éste trabajo se fomenta en la creación de una plataforma de bajo coste capaz de auditar de forma concisa el registro de actividades generado por elementos de seguridad de una red. Se ha buscado un producto a medida de los requerimientos planteados por de El cliente: Un producto que satisficiera las necesidades requeridas,, que dispusieran de soporte personalizado y que pudiesen tomar decisiones a lo largo del desarrollo, siendo características que no puede contemplar un producto corporativo de venta a gran escala, totalmente implementado y con soporte generalizado.

Por otro lado, el objetivo de Linecom, empresa que desarrolla la plataforma, era la obtención de una aplicación de formato genérico, utilizando librerías de código abierto y manteniendo la posibilidad de añadir nuevos tipos de logs a la plataforma, pudiendo acoplar el producto a las necesidades de no uno sino diversos clientes.

Las diferentes necesidades de El cliente crearon la base del proyecto y los designios de Linecom un camino específico de desarrollo. Los retos y soluciones fundamentales son los descritos a continuación:

- El sistema había de ser independiente del sistema operativo en el cual corriese. Por ello se escogió una plataforma de desarrollo Java, de lenguaje interpretado, ya que el código de ejecución siempre es el mismo y convertido a ordenes nativas del sistema operativo. Este

- aspecto no solo afecto al desarrollo de la plataforma sino a la elección de los elementos complementarios, como agentes y base de datos que tenían que aceptar el mayor número de plataformas posible para su ejecución.

- El coste de la aplicación tenía que ser controlado ya que es un aspecto fundamental para competir con los productos corporativos específicos para el seguimiento de logs, por ello se utilizaron librerías y recursos de código abierto, como JasperReports, Agente SNARE, base de datos MySQL y plataforma de desarrollo/web Java/Tomcat.

- Recopilar y ordenar los eventos generados por los sistemas operativos era el objetivo primordial y base de la plataforma. En principio la aplicación solo había de ser capaz de auditar eventos de seguridad de los sistemas operativos, dejando abierta la posibilidad de auditar cualquier otro registro de otro elemento de seguridad. Se instalaron agentes capaces de extraer los eventos en tiempo real y se implementó un servicio capaz de recibir esos eventos y guardarlos de forma temporal. Mediante la información recopilada por el servicio, el sistema era capaz de leer y extraer los datos críticos, formatearlos para hacerlos clarificadores y concisos y guardarlos de forma ordenada en la base de datos para poder acceder a ellos de una forma ágil y segura.

- El reporte de la información era la operación básica que generaba los resultados de la plataforma. Éste estuvo basado en la información contenida en la base de datos. A través de una página web con tecnología JSP el usuario, de forma transparente, lanza los procesos Java mediante simples formularios http. Para no entorpecer el procesamiento de datos con la lógica web se instanció un sistema basado de diferentes hilos de ejecución independientes, mejorando así a eficiencia de la aplicación.

- El motor de búsqueda que facilitara la obtención de eventos de manera detallada se implementó basándose en la información almacenada en la base de datos. Con una filosofía similar a la del núcleo de la plataforma, es decir una ejecución multi-hilo, y controlando la apertura de las conexiones a la BD se implementó el motor, reflejando los resultados en el mismo navegador mediante código http.
-
- La capacidad de inclusión de nuevos logs de diferentes tecnologías resultó ser un aspecto difícil de asimilar ya que como hemos mencionado anteriormente el producto se fabricó a medida. Ciertos aspectos, como el núcleo de “parseo” y las plantillas de los reportes están íntimamente ligadas al contenido de los logs, es decir, que por cada log con formato diferente hay que implementar nuevo código para poder adecuar la plataforma al contenido de entrada.

Los inconvenientes de la plataforma resultaron estar asociados a la gran cantidad de volumen de datos a tratar. En un principio la plataforma solo recibía logs mensuales y debía de generar los consiguientes documentos. Una vez

El cliente decidió implementar el sistema de recogida de Linecom, la cantidad de datos se redujo drásticamente ya que los logs serían escogidos en el agente antes de ser enviados al servidor central y éstos serían tratados de forma diaria. Aunque la plataforma se convirtió en un proceso automático realizado de forma periódica aun, la necesidad de manejo de grandes de parámetros de entrada ha sido necesaria debido al proceso de reporte. Es en este aspecto donde la plataforma se puede mejorar, separando el proceso de tratado del motor de búsqueda y utilidad de reporte, ya que la tecnología JSP es menos eficiente en ejecución que un servidor de aplicaciones o una ejecución directa sobre JVM.

Los objetivos de mejora de la plataforma por parte Linecom son enfocados a dotarla de manejabilidad:

- Incluir un apartado gráfico para el mantenimiento de la base de datos de logs, pudiendo ver, borrar, crear y editar cualquier tabla contenida. Utilizando la misma lógica de visualización que para el motor gráfico e implementando formularios asociados a las filas de las tablas expuestas seriamos capaces de realizar esta función mediante el mismo navegador.
- Posibilidad de configuración de los agentes de forma centralizada. Para el agente Snare, se deberían de introducir los parámetros de seguridad para entrar en el agente, en el caso de Unix se debería de poder editar el fichero de configuración de Syslog de forma remota a través del navegador.
- Ya que la plataforma es centralizada, dotarla de un sistema de control de acceso, vía Tomcat ya implementado por la compañía sustentadora de la plataforma. El proceso de autenticación puede ser realizado mediante el archivo local users-tomcat.xml o a través de canales encriptados con la base de datos.

Si bien es cierto que se han cumplido todos los objetivos demandados por la empresa, una aplicación de éstas características ha de estar en constante mantenimiento para prevenir y/o solucionar futuros problemas de incompatibilidades y mejoras de rendimiento ya que es un producto realizado “a medida” para una compañía determinada.

BIBLIOGRAFÍA

[1] J. Brittain y F. Darwin, *Tomcat: the definitive guide,* O'Reilly, cop. 2008.

[2] B. Kurniawan, *Java for the Web with Servlets, JSP, and EJB,* Indianapolis New Riders Publishing, 2002.

[3] W. Da Silva, JSP and tag libraries for web development, Indianapolis New Riders, 2001.

[4] B. Schwartz, *High performance MySQL,* Cambridge: O'Reilly, cop. 2008.

Artículos de gestión de logs:

[5] http://www.borrmart.es/articulo_redseguridad.php?id=1369&numero=27

[6] http://www.borrmart.es/articulo_redseguridad.php?id=1009&numero=22

Artículos de soporte de Microsoft:

[7] http://support.microsoft.com/kb/308427

[8] http://www.microsoft.com/technet/support/ee/SearchResults.aspx

[9] http://msdn.microsoft.com/en-us/library/aa964766(vs.85).aspx

[10] http://msdn.microsoft.com/en-us/library/aa363662(VS.85).aspx

Manual de utilización de Tomcat v5.5:

[11] http://tomcat.apache.org/tomcat-5.5-doc/index.html

Tutoriales de Java:

[12] http://java.sun.com/javase/reference/tutorials.jsp

Manual de utilización de MySQL 5.1:

[13] http://dev.mysql.com/doc/refman/5.1/en/

Tutorial de uso de librerías JasperReports:

[14] http://jasperforge.org/website/jasperreportswebsite/trunk/tutorial.html?group_id=252